DBSJ Journal, Vol.11, No.2
October 2012

XPathを用いた暗号化XML文書検索手法の提案

Searching Encrypted XML Documents by Using XPath

中村伸一♥ 山本博章♦

Shin-ichi NAKAMURA♥ Hiroaki YAMAMOTO♦

DaaS(Database-as-a-Service)における情報セキュリティ上の重要な問題の1つは，データへのアクセス権限のない管理者や第3者などによって，データベースを構成するファイルが不正に持ち出されることである．そのため，検索内容や結果を暗号化したままで情報検索する手法が提案されている．本研究は，XMLデータに焦点を当て，与えられたXPathに対し，暗号化されたXML文書からそのXPathと一致する部分木を検索するための手法を提案する．従来の手法はXML文書の構造を利用するため，データベースの管理者は構造を知ることができたが，我々の手法は検索内容や結果だけでなく構造も隠す．したがって，権限のない管理者はXMLの内容だけでなく構造も知ることができない．本稿では，提案手法の安全性について議論し，さらに，実験的に検索性能についても評価する．

One of the most important problems in the security of DaaS (Database-as-a-Service) is that files stored in a database are illegally accessed by the database administrator and unauthorized people who do not have any permission to access the contents of the database. The effective way to protect data stored in a database is to encrypt it. Therefore, efficient search methods on encrypted data have been studied. In this paper, we focus on XML, and propose a method which searches for subtrees matching a given XPath on encrypted XML documents. Since the existing search methods make use of the structure of XML documents, an administrator can know the structure. However, the proposed method encrypts not only contents of XML documents but also the structure. Hence, an administrator can know neither the contents nor the structure of XML documents. We discuss security on the proposed method, and also evaluate the performance by some experiments.

1. はじめに

近年，インターネット回線の高速化と信頼性の向上により，Amazon SimpleDB[1]のような，インターネットを経由してデータベースの利用環境を提供するサービス(Database as a Service(以下DaaS))が始まっている．DaaSは，専門の管理者が複数のサーバを集約して運用する形態をとっているため，必要な技術者の雇用や情報機器が不要となり，データベースの運用費用の削減や利用までの時間短縮が期待できる．

しかし，DaaSに関する情報セキュリティ上の問題として，管理者の違法な目的や不十分な管理から，サーバからデータベースを構成するファイルが持ち出される問題がある．解決策として，ファイルの暗号化が挙げられるが，データベースの処理において暗号化されたデータがサーバ上で復号されるため，管理するサーバの権限を利用してデータベースを操作することで，管理者がファイルを持ち出すことが可能である．そのため，検索内容や結果を暗号化したままでデータを検索する手法が研究されている．

検索内容や結果を暗号化したままでデータを検索する手法の1つとして，Brinkman[2]らはXML文書を検索する手法を提案している．Brinkman[2]らの手法は，暗号化したままでXML文書の要素名や属性を検索するが，検索時にXML文書の構造を利用する．そのため，XML文書の構造の暗号化を行っていない．したがって，DaaS上で公開や販売されているXML文書の検索をBrinkman[2]の手法で運用する場合，管理者が同じXML文書を入手できると，構造を利用して検索内容や結果が推測される問題がある．例えば，公開や販売されているXML文書が辞書の場合，管理者に検索している単語だけでなく，単語を検索している文書が推測される問題がある．

本論文では，この問題に対応するため，管理者が検索内容や結果だけでなく構造もわからない，安全な暗号化XML文書検索手法を提案する．

提案手法は，事前にXML文書を暗号化して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の一部の要素と構造が同じ部分木のデータを暗号化したまま取得し，取得したデータを復号してXPathとマッチするか判定する．公開や販売されているXML文書をDaaSで運用する場合に提案手法は有効であり，DaaSで安全に利用できるXML文書の範囲を広げることができる．Brinkman[2]との性能比較から，提案手法が対応するXPathの問合せ範囲において，概ね速度面でも優位である．

本論文は次のように構成されている．1章のまえがきに続き，2章は関連研究について述べる．3章は提案手法について前提となる知識について述べる．4章は従来の暗号化XML文書検索の問題点について述べる．5章は提案手法を利用した暗号化XML文書検索システムと具体的なアルゴリズムについて述べ，また，提案手法の安全性について議論する．6章は5章で述べた暗号化XML文書検索システムによる実験と考察について述べる．7章はまとめと今後の課題について述べる．

2. 関連研究

2.1 XML文書全体を暗号化する検索手法

Brinkman[2]らは，Song[3]らが提案した暗号化された文字列を検索する手法を，XML文書の検索に拡張した手法を提案している．中村[4]らは，RDBに暗号化したXML文書を登録し，検索するXML文書の根と同じ要素名を根とする部分木をRDBから検索し，クライアント側で復号して検索するXML文書とマッチングすることで，検索する手法を提案している．Yang[5]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に暗号化したXML文書のデータを登録し，検索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という手法を提案している．Schrefl[6]らは，利用者側には文書の構造に関する情報を持たせ，サーバには各ノードへのパス情報を暗号化して作成し

♥学生会員 信州大学大学院総合工学系研究科
s09t203@shinshu-u.ac.jp
♦信州大学工学部
yamamoto@cs.shinshu-u.ac.jp

たインデックスと各ノードの要素名を暗号化して作成したインデックスのみを持たせる．検索は，利用者とサーバとの間で暗号化したパス情報や要素名をやり取りすることによって行い，最終結果の構造を利用者側でチェックする手法を提案している．

2.1 XML 文書の一部を暗号化する検索手法

Lee[7]らは，RDB に XML 文書の要素名と復号に必要な鍵と XML 文書の要素の位置を暗号化したインデックスを登録し，RDB 側で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提案している．Jammalamadaka[8]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提案している．Wang[9]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，暗号化した要素名と DSI のインデックスと 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提案している．Yang[5]や Schrefl[6]の手法は，利用者側にインデックスを持つため，インデックスを保存するためのメモリ等の負担が利用者側にかかるが，提案手法は利用者側にインデックスを持たないため，Yang[5]や Schrefl[6]の手法に比べて利用者側への負担は少ない．

3. 準備

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，木の各ノードは要素名でラベル付けされている．木の各ノードの深さは，枝の長さを 1 とするとき，根からノードまでの枝数である．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 1 (a)を同等の木で表現したものと葉を図 1 (b)に示す．

提案手法は，XML 文書に対し，それが表す木の各ノードに位置に関するラベリングを行い，その情報を RDB に保存する．ここで，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．提案手法は，リージョン[10]と呼ばれるデータ構造と深さとパスを各ノードにラベリングする．リージョンは，木の各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の中で一番大きな位置番号とする．リージョンは，木のノードの位置だけでなく，ノード間の関係(先祖，子孫，親子，系統)も，把握することができる．

各ノードをラベリング(start, end, 深さ，パス)した木を図 1(b)に示す．パスのラベリングは，括弧内に左から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．

XPath[11]は，XML 文書の特定の部分を指定するため，W3C が標準化を行った言語である．XPath は一般的にロケーションパスで表現される．ロケーションパスは 1 つまたは複数のロケーションステップの並びとして記述される．

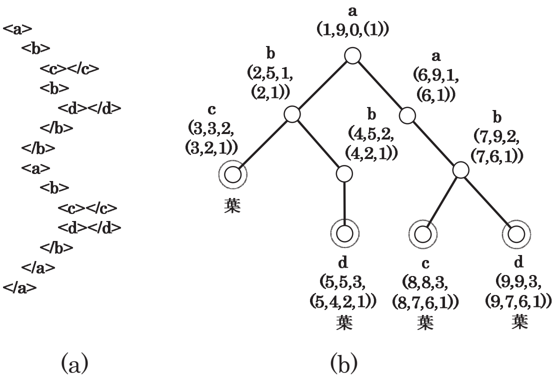


図 1 木構造

Fig.1 Tree Structure

4. 従来の暗号化 XML 文書検索の問題点

Brinkman[2]の手法は XML 文書の構造の暗号化を行っていない．そのため，RDB に登録している XML 文書と同じ XML 文書を入手できれば，RDB に登録している XML 文書と同じ位置に同じ要素があることから，以下の攻撃手法で，検索内容や結果が推測される問題がある(図 2)．

(1) 攻撃者が RDB に登録している XML 文書を入手する．
(2) 利用者が検索している要素の位置を攻撃者が入手する．
(3) 攻撃者が(1)で入手した XML 文書の(2)で入手した位置にある要素を参照することで要素名を推測する．

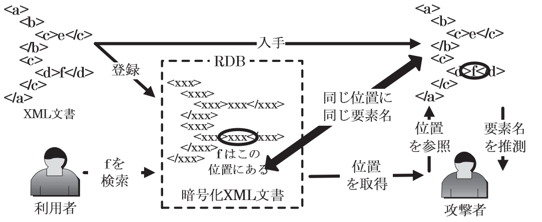


図 2 攻撃手法

Fig.2 Mode of Attack

Brinkman[2]の手法で構築した英和辞書の検索サービスがこの攻撃手法で攻撃された場合，辞書を検索する単語が攻撃者に推測される直接的な被害だけでなく，翻訳する文章も攻撃者に推測される間接的な被害も発生する(図 3)．

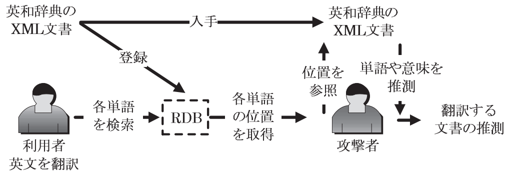


図 3 検索サービスへの攻撃

Fig.3 Attack to Search Service

5. 暗号化 XML 文書検索システム

4.章で述べた攻撃手法に対応した，XML 文書の構造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．

5.1 概要

システムの概要を図4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，暗号化したデータを RDB サーバに登録する．検索は，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 5.4 節の手法に基づき，検索結果を返す．なお，RDB サーバからクライアントがデータを取得した場合，K によるデータの復号を行っている．

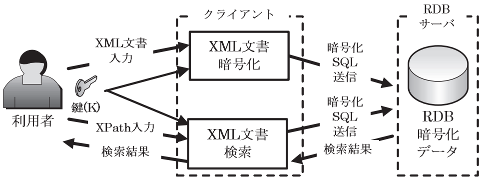


図 4 暗号化 XML 文書検索システムの構成

Fig.4 Configuration of Encrypted XML Document Search System

5.2 暗号化 XML 検索手法の暗号化範囲の比較

提案手法と論文[2]と[4]～[9]が XML 文書を暗号化する範囲を表 1 に示す．範囲は XML 文書を構成する要素と属性と内容と構造とする．暗号化範囲について，○はすべて暗号化，△は一部を暗号化，×は暗号化しない，斜線は検索が対応しないことを示す．

提案手法は，表 1 から XML 文書の要素と構造に関して暗号化したままで検索する．

表 1 暗号化範囲

Table 1 Degree of Protection

論文	暗号化範囲			
	要素	属性	内容	構造
Brinkman[2]	○	○	／	×
中村[4]	○	／	／	○
Yang[5]	○	○	○	○
Schrefl[6]	○	○	○	○
Lee[7]	△	／	△	△
Jammalamadaka[8]	△	／	△	△
Wang[9]	△	／	△	×
提案手法	○	／	／	○

5.3 適用範囲

このシステムの適用範囲を以下に定める．

- XML文書のRDBサーバへの登録と検索は同じKを使用する．
- システムが扱うXPathの構文は図5で示すEBNFで記述した範囲とする．<Tag>はXPathにおける要素名を指す．分岐点はXPathにおける2つ以上の子要素を持つ要素の中で最も根に近い要素を指し，XPathに高々1つ存在する．
- XML文書の更新は行わない．

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

図 5 システムが扱う XPath の構文

Fig.5 XPath Supported by Our System

5.4 提案手法

5.4.1 概要

提案手法は，XPath の分岐点と葉の要素名を取得し(図 6(a))，分岐点と葉の要素名と同じ要素を XML 文書から検索する(図 6(b))．次に，分岐点の要素名と同じ要素を根とした部分木の中に，すべての葉の要素名がある部分木のデータを XML 文書から検索する(図 6(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 6(d))．XPath における分岐点は 5.3 節で述べたように高々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．

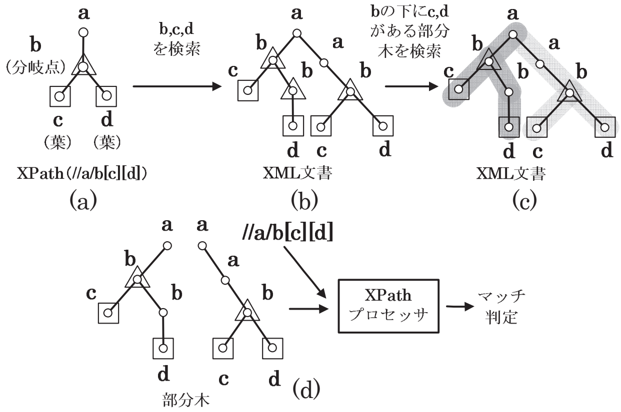


図 6 提案手法

Fig.6 Proposed Method

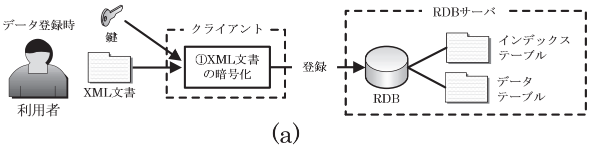
システムは，クライアントと RDB サーバとの間で以下のデータのやり取りを行う．

(1) 要素名等の情報が暗号化されたデータテーブル，同じ要素名の位置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 7(a)①)．

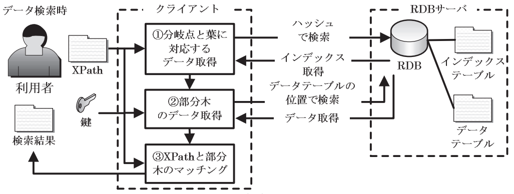
(2) XPath とマッチングする部分木のデータを取得するため，XPath から分岐点と葉の要素名を取得し，インデックステーブルとデータテーブルを利用して，分岐点と葉の要素名と同じ要素のデータを取得する(図 7(b)①)．

(3) (2)で取得したデータで分岐点の要素名と同じ要素について，これを根とする部分木の中に，葉と同じ要素名がすべてあるか調べ，あればデータテーブルから根から葉までのパス上の要素のデータを取得する(図 7(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 7(b)③)．



(a)



(b)

図 7 暗号化 XML 文書検索システムの構成

Fig.7 Configuration of Encrypted XML Document Search System

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，暗号化する平文のデータをD，ランダム文字列を S，鍵をK，E_{enc}を暗号化関数，D′ を暗号化データとして式(1)，復号はE_{dec}を復号関数として式(2)に従って行っている．

$$D' = E_{enc}(K, D, S) \qquad (1)$$
$$D = E_{dec}(K, D') \qquad (2)$$

5.4.2 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 1(a)を XML 文書とする DTBL と ITBL の構造を図 8 に示す．

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，Data は 5.4.1 節で述べた手法で暗号化された XML 文書の要素の情報(要素名，start，end，深さ，パス)である．DTBL のパスについて，3 章で述べたパスのラベリングと異なり，要素の start ではなくパスの各要素に対応する DTBL の Loc となる．

ITBL の項目について，Hash は K で暗号化しハッシュ化した要素名，Data は 5.4.1 節で述べた手法で暗号化された同じ Hash である要素の情報の集合 (($start_1$, end_1, Loc_1), ... , ($start_i$, end_i, Loc_i)) (i=Hash が同じ要素数)である．ITBL の Loc は要素に対応する DTBL の Loc である．図 8 について，DTBL と ITBL の Data の括弧部分は，平文の Data である．紙面の関係から DTBL と ITBL の Data の暗号列の途中を省略する．この処理における計算量は，XML 文書の要素数を n とした場合 *O* (n)である．

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

ITBL

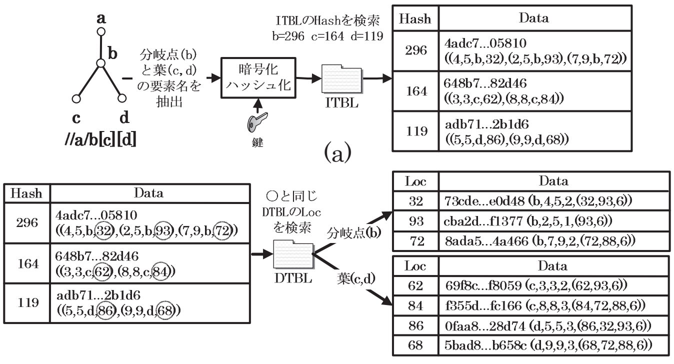
図 8 図 1(a)を XML 文書とする DTBL と ITBL の構造

Fig.8 Structure of DTBL and ITBL for the XML Document in Fig. 1(a)

5.4.3 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データを取得する．この処理における計算量は，XPath の葉の要素数を n とした場合 *O* (n)である．

(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 9(a))．
(2) 取得した ITBL の Loc (図 9 ○)と DTBL の Loc が同じデータを取得する(図 9(b))．



Hash	Data
296	4adc7...05810 ((4,5,b,32),(2,5,b,93),(7,9,b,72))
164	648b7...82d46 ((3,3,c,62),(8,8,c,84))
119	adb71...2b1d6 ((5,5,d,86),(9,9,d,68))

(a)

Hash	Data
296	4adc7...05810 ((4,5,b,32),(2,5,b,93),(7,9,b,72))
164	648b7...82d46 ((3,3,c,62),(8,8,c,84))
119	adb71...2b1d6 ((5,5,d,86),(9,9,d,68))

Loc	Data
32	73cde...e0d48 (b,4,5,2,(32,93,6))
93	cba2d...f1377 (b,2,5,1,(93,6))
72	8ada5...4a466 (b,7,9,2,(72,88,6))

Loc	Data
62	69f8c...f8059 (c,3,3,2,(62,93,6))
84	f355d...fc166 (c,8,8,3,(84,72,88,6))
86	0faa8...28d74 (d,5,5,3,(86,32,93,6))
68	5bad8...b658c (d,9,9,3,(68,72,88,6))

(b)

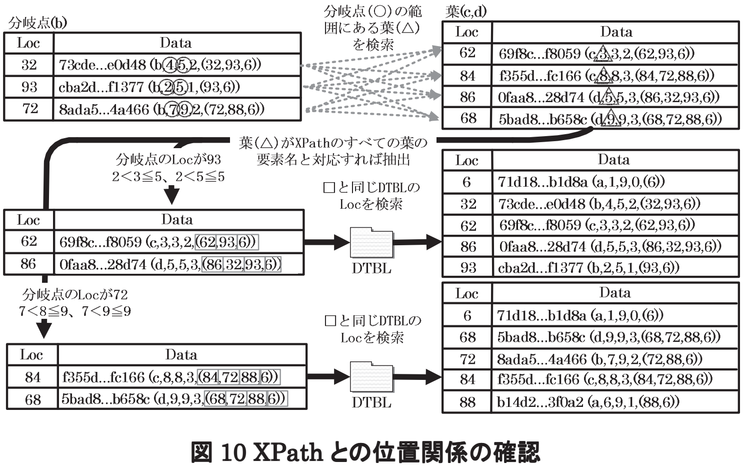
図 9 XPath の要素名の取得

Fig.9 Obtaining XPath Element Name

5.4.4 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，5.4.3 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 10)．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データのパスを部分木のデータとして取得する．この処理における計算量は，5.4.3 節で取得した分岐点のデータ数を m，葉のデータ数を n とした場合 *O*(m×n)である．

(1) 分岐点の start と end (図 10 ○)の間に葉の start (図 10 △)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 10 □)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．



分岐点(b)

Loc	Data
32	73cde...e0d48 (b,4,5,2,(32,93,6))
93	cba2d...f1377 (b,2,5,1,(93,6))
72	8ada5...4a466 (b,7,9,2,(72,88,6))

葉(c,d)

Loc	Data
62	69f8c...f8059 (c,3,3,2,(62,93,6))
84	f355d...fc166 (c,8,8,3,(84,72,88,6))
86	0faa8...28d74 (d,5,5,3,(86,32,93,6))
68	5bad8...b658c (d,9,9,3,(68,72,88,6))

Loc	Data
62	69f8c...f8059 (c,3,3,2,(62,93,6))
86	0faa8...28d74 (d,5,5,3,(86,32,93,6))

Loc	Data
6	71d18...b1d8a (a,1,9,0,(6))
32	73cde...e0d48 (b,4,5,2,(32,93,6))
62	69f8c...f8059 (c,3,3,2,(62,93,6))
86	0faa8...28d74 (d,5,5,3,(86,32,93,6))
93	cba2d...f1377 (b,2,5,1,(93,6))

Loc	Data
84	f355d...fc166 (c,8,8,3,(84,72,88,6))
68	5bad8...b658c (d,9,9,3,(68,72,88,6))

Loc	Data
6	71d18...b1d8a (a,1,9,0,(6))
68	5bad8...b658c (d,9,9,3,(68,72,88,6))
72	8ada5...4a466 (b,7,9,2,(72,88,6))
84	f355d...fc166 (c,8,8,3,(84,72,88,6))
88	b14d2...3f0a2 (a,6,9,1,(88,6))

図 10 XPath との位置関係の確認

Fig.10 Verifying the Positional Relationship with XPath

5.4.5 XPath と部分木のマッチング

5.4.4 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 11)．この処理における計算量は，利用する XPath プロセッサに依存する．

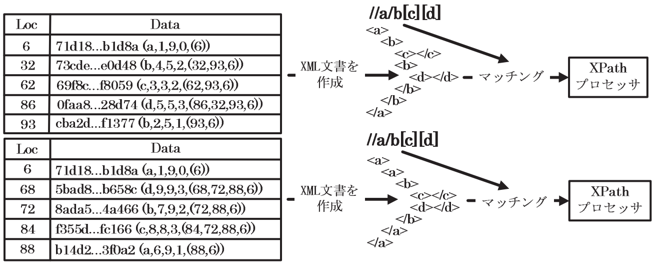


図 11 XPath プロセッサによるマッチング

Fig.11 Matching by XPath Processor

5.5 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

- DTBL と ITBL の Data に対する既知平文攻撃
  攻撃を行うには，要素に関する情報がある平文が必要だが，5.4.1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．
- DTBL と ITBL の Data に対する選択平文攻撃
  既知平文攻撃で述べたように，ランダムな文字列が追加して暗号化してあるため，平文や平文に対する暗号文も入手することができない．そのため，攻撃はできない．
- ITBL の Hash に対する攻撃
  要素名を暗号化してハッシュ関数で処理したものであるため，要素名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

- 既知平文攻撃と選択平文攻撃
  構造に関する情報が DTBL と ITBL の Data に含まれているため，(1) 要素に対する攻撃の既知平文攻撃と同じ理由から攻撃はできない．
- 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) 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，攻撃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．

5.6 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2点の改良を行っている．6章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，5.3節で述べたように XML 文書は修正されないため，一度メモリに記憶したデータの更新は行わない．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) 5.4.4節の処理に Stack-Join-Desc[12]を利用することにより，計算量を5.4.3節で取得した分岐点のデータ数をm，葉のデータ数をn，5.4.4節で抽出した葉と対応する分岐点のデータ数をoとした場合 $O(m+n+o)$ とする改良を行うことができる．

6. 実験

6.1 実験環境について

提案手法と Brinkman[2]の手法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 2GB，CentOS 6.0，クライアントは，Intel Core2Duo 2.26GHz，メモリ 2GB，Windows XP 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 5.0.22 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[2]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[13]を採用した．提案手法の実験は 5 章で述べたシステムを使用した．

XML 文書は XMark[14]で公開されている XML 文書作成ソフトウェアで，100MB の XML 文書を作成した．XML 文書の要素数は 2,858,801 個であった．検索する XPath は表 2 で示す．

XML 文書作成ソフトウェアは，オークションに関するデータを想定した XML 文書を作成する．引数の指定により XML 文書の大きさも指定できる．図 12 に作成する XML 文書の構造の概要を示す．提案手法はこの XML 文書の要素名と構造を暗号化したままで検索する．

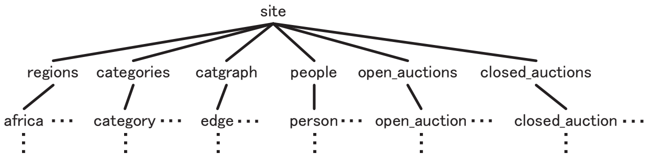


図 12 XML 文書の構造

Fig.12 Element Hierarchy of XML Document

実験は，提案手法と Brinkman[2]の手法の検索時間に関する測定を行い，結果は 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[2]の手法は対応する 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対応する XPath の範囲内での評価となる．

表 3 に提案手法と Brinkman[2]の手法が対応する 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 は対応する軸の名称，Node Test は対応するノードの指定範囲，Predicate は対応するデータ型(DataType)や演算子(Operator)や関数(Function)を示す．×は対応しないことを示す．表 3 の①は XPath の先頭のみ対応，②は子要素のみ対応することを示す．

表 2 検索する XPath

Table 2 XPath to Search

Query	XPath
Q1	/site/regions/europe/item/description
Q2	/site/people/person[name][emailaddress][phone]
Q3	//closed_auction/annotation[author][description][happiness]
Q4	//description/text

表 3 Brinkman[2]と提案手法が対応する XPath の比較

Table 3 Comparison of XPath Supported by Brinkman et al.'s Method [2] and the Proposed Method

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	×	×

6.2 実験結果

表 4 に提案手法と Brinkman[2]の手法の検索時間を示す．項目について，Brinkman は Brinkman [2]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，Proposed の RDB Server は提案手法の RDB サーバの実行時間，Proposed の Total は提案手法全体（クライアントと RDB サーバの合計）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 4 について，Brinkman[2]の手法と提案手法を比較すると，Q1 を除いて提案手法の方が約 1.1～4 倍速い．この理由として，Brinkman [2]の手法は，XML 文書のすべての要素を対象に，XPath にマッチする要素を選択することで検索する．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 [2]の手法に比べて，RDB サーバが処理する要素数が少ないため速いと考えられる．

Q1 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を持つ要素のパスを部分木として取得する．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が XML 文書中に多い場合，Brinkman[2]に比べて RDB サーバが処理する要素数が多くなり，Brinkman[2]より遅くなったと考えられる．

表 4 実験結果(検索時間)

Table 4 Test Results for Search Times

Query	Brinkman (ms)	Proposed (ms)			Matches
		Client	RDB Server	Total	
Q1	17,245	98,259	12,925	111,184	6,000
Q2	90,735	47,328	7,258	54,586	12,679
Q3	439,615	84,758	14,352	99,110	9,750
Q4	431,247	328,894	41,754	370,648	31,368

7. おわりに

本論文では，管理者が検索内容や結果だけでなく構造もわからない，安全な暗号化 XML 文書検索手法を提案した．

提案手法について，XML 文書を暗号化する範囲の攻撃に対して安全性の評価を行い安全であることを示した．XMark[14]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[2]の手法との性能比較を行い，提案手法が対応する XPath の問い合わせ範囲において，一部の Q(Q1)以外は高速である結果となった．

XPath の問い合わせ範囲の拡張や XML 文書の更新に対応した検索手法の提案が今後の課題である．

【文献】

[1] "AmazonSimpleDB", http://aws.amazon.com/jp/simpledb/.

[2] 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).

[3] 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).

[4] 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", FIT2008 , pp.43–44 (2008).

[5] 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).

[6] 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).

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

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

[9] 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).

[10] 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).

[11] "XML Path Language (XPath) 2.0 (Second Edition)", http://www.w3.org/TR/xpath20/.

[12] 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).

[13] "XML::XPath - search.cpan.org", http://search.cpan.org/~msergeant/XML-XPath-1.13/.

[14] "XMark - An XML Benchmark Project", http://www.xml-benchmark.org/.

中村伸一 Shin-ichi NAKAMURA

信州大学大学院総合工学系研究科博士課程在学中．信州大学大学院工学系研究科・情報工学専攻修了．データベースに関する研究に従事．日本データベース学会学生会員．

山本博章 Hiroaki YAMAMOTO

信州大学工学部教授．東北大学大学院工学研究科博士課程修了．工学博士．アルゴリズムの開発，オートマトン，情報検索に関する研究に従事．電子情報通信学会，情報処理学会，日本ソフトウェア科学会，EATCS 各会員．