Vol.2011-CSEC-53 No.20
Vol.2011-IOT-13 No.20
2011/5/13

ブルームフィルタを用いた検索自由度の高い検索可能暗号の設計と実装評価

菅 孝徳 [†1] 西出 隆志 [†2] 櫻井 幸一 [†2]

データの暗号化と検索を両立可能な検索可能暗号と呼ばれる暗号がある．しかし，共通鍵暗号方式の検索可能暗号において，柔軟な検索を行うためには，暗号化を行う際に考えうる全てのキーワードを列挙しなければならず，キーワードに相当する暗号文のデータ量が大きくなってしまっていた．そこで本研究では，暗号化の際に考えうる全てのキーワードを列挙することなく，指定された位置の文字が何であるかを判別する論理式を用いて柔軟なクエリを構成可能な検索可能暗号を提案する．本提案方式では，クライアントで誤りを検出可能であることから，ブルームフィルタを用いてある程度の偽陽性を認めることでデータ効率を高めている．更に，本提案方式を実装することで，現実的な時間で検索処理を行うことが出来る事を示す．

Design, Implementation and Evaluation of Symmetric Key Encription with Flexible Keyword Search by Using Bloom Filter

Takanori Suga [†1], Takashi Nishide [†2] and Kouichi Sakurai [†2]

There are encryption schemes which enable both encryption and search. However, in traditional symmetric key searchable encryption schemes, we must enumerate all considerable keywords in encryption. Therefore, we must have large amount of ciphertext which corresponds to keywords if we need flexible search. In this work, we propose a symmetric searchable encryption scheme which allows users to construct flexible queries. This scheme does not require enumeration of all considerable keywords and enable flexible search with boolean expression. This scheme utilizes Bloom filter to reduce a size of ciphertexts. Bloom filter has some probability of a false positive. However, it does not have any problem in practical use because a client can verify the result. We show this scheme can search in practical time.

1. はじめに

1.1 研究背景

近年，個人情報等の機密情報の漏洩事件が問題となっている．2009 年には，約 572 万人の個人情報が漏洩しており，その想定損害賠償総額は約 3,890 億円にも上る[9]．今後，クラウドコンピューティングの普及が進むと，外部へのデータ保存が進み，この問題は更に深刻化すると予想される．情報漏洩を防ぐ対策として機密データの暗号化を行う場合，計算機から内部に含まれるデータを知ることが出来ないため，一般に，検索などの処理を行うことが出来なくなる．しかし，データを暗号化しながら，そのデータに対する検索を可能とする，検索可能暗号と呼ばれる暗号が存在する．

2000 年に初めての検索可能暗号が提案されて以来，様々な検索可能暗号が提案されている．しかし，既存の共通鍵暗号方式の検索可能暗号においては，サーバ側でキーワードの完全一致しか行うことが出来ない為，キーワードを暗号化する際に考えうる全てのキーワードを列挙しなければならず，柔軟な検索を行うためには暗号文のデータ量が大きくなってしまっていた．

本研究では，データ所有者がサーバにデータを送信する際に，考えうる全てのキーワードを送信することなく，柔軟な検索を可能とする共通鍵暗号方式の検索可能暗号を提案する．この方式を SEFKS(Symmetric-key Encryption with Flexible Keyword Search) と呼ぶこととする．SEFKS では，何文字目に何という文字があるかを指定することで検索を行うことができる．また，SEFKS はブルームフィルタを用いて，データ量を削減している．更に，SEFKS を実装することで，速度，データ量のどちらも実用的な範囲に収まっていることを示す．

1.2 既存方式

2000 年に Song らによって，最初の実用的な共通鍵暗号方式の検索可能暗号が提案された[8]．この方式では，完全一致検索のみが利用可能であり，柔軟な検索を行うことはできない．柔軟な検索を目的として提案された方式としては，2010 年に Li らによって提案された

†1 九州大学大学院システム情報科学科情報学専攻
Department of Informatics, Graduate School of Information Science and Electrical Engineering, Kyushu University
†2 九州大学大学院システム情報科学研究院情報学部門
Department of Informatics, Faculty of Information Science and Electrical Engineering, Kyushu University

© 2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.20
Vol.2011-IOT-13 No.20
2011/5/13

類似検索を行うことの出来る検索可能暗号方式がある[5]．この方式は，データを登録する側とデータを検索する側の両方でキーワードを列挙することで，実現されている．その為，更に柔軟な検索を行うにはインデックス，トラップドアの両方のデータ量，及びその生成にかかる計算量が増大する．

一方で，公開鍵暗号方式の検索可能暗号としては 2004 年に初めて Boneh らによって提案されている[2]．また，柔軟な検索を目的として提案された方式としては，2010 年に吉田らによって提案された部分一致検索を行うことの出来る検索可能暗号方式がある[10]．吉田らの方式では本提案方式と同様のクエリが構成出来るが，公開鍵暗号を必要としない環境では共通鍵暗号方式の方が高速に計算可能である為，共通鍵暗号方式を用いる方が望ましい．しかし，そのような共通鍵暗号方式の検索可能暗号は今まで存在していなかった．

2. 検索可能暗号

2.1 検索可能暗号の概略

検索可能暗号の利用は，以下のように行う．図 1 に概略図を示す．

(1) データの所有者と，検索を行うユーザの間で，事前に鍵の共有を行っておく．
(2) データの所有者は，暗号化したデータの内容を表すキーワード Kw_A を指定する．このキーワードは複数指定することも出来る．
(3) Kw_A のトラップドア Td_A を計算する．
(4) 暗号化したデータ C_A と Td_A をサーバに送信する．
(5) サーバは C_A と Td_A をデータベースに登録する．
(6) データの検索を行うユーザは，検索したいキーワード Kw_B のトラップドア Td_B を計算する．
(7) Td_B をサーバに送信する．
(8) サーバは，データベースから Td_B にヒットするデータを検索し，検索を行ったユーザに結果を返却する．

サーバは，Td_A, Td_B から Kw_A, Kw_B を知ることは出来ない．検索可能暗号では，このようにサーバにさえキーワードを知らせることなく，検索することが出来る．

2.2 検索の種類

検索方式として，以下のようなものがある．

完全一致検索 完全に一致するものがヒットするような検索を完全一致検索という．

前方一致検索 検索するキーワードを先頭に含むものがヒットするような検索を前方一致検索という．

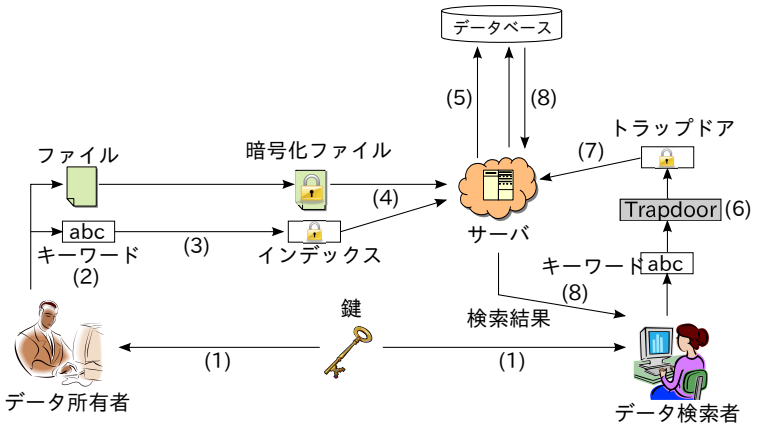

図 1 検索可能暗号の概略図
Fig. 1 Schema of Searchable Encryption

後方一致検索 検索するキーワードを末尾に含むものがヒットするような検索を後方一致検索という．

部分一致検索 検索するキーワードを任意の場所に含むものがヒットするような検索を部分一致検索という．

類似検索 検索するキーワードと一定以内の編集距離であるようなキーワードを持つものがヒットするような検索を類似検索という．

本論文における柔軟な検索とは，これら全ての検索が可能なことを言うものとする．

3. 要 素 技 術

3.1 ハッシュ関数

ハッシュ関数とは，ある入力に対してその入力を表すある数値を得る関数のことである[3]．ハッシュ関数は，誤り訂正や連想配列，暗号化，認証などに用いられるが，暗号に用いる場合は用途に応じて以下のような安全性を持つハッシュ関数を利用する必要がある．

原像計算困難性 ハッシュ $y = h(m)$ が与えられた時に，m を求めることが困難であるような性質．

第 2 原像計算困難性 入力 m_1 が与えられた時，$h(m_1) = h(m_2)$ となるような別の入力

© 2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.20
Vol.2011-IOT-13 No.20
2011/5/13

m_2 を見つけることが困難であるような性質.

衝突困難性 $h(m_1) = h(m_2)$ となるような異なる m_1, m_2 を見つけることが困難であるような性質.

3.2 ブルームフィルタ

Bloom は，1970 年に誤りを許すことで，効率的に空間を利用することのできる，確率的なデータ構造を提案した[1]．これはブルームフィルタと呼ばれている．

ブルームフィルタにおいて，ある確率で要素が含まれないにも関わらず，要素が含まれていると判定してしまう偽陽性が存在する．一方で，要素が含まれているにも関わらず，要素が含まれていないと判定してしまう偽陰性は存在しない．

ブルームフィルタの構成

ブルームフィルタは，0 から N - 1 までのアドレスを持つ，N 個のビットの集合である．初めに，この集合は全て 0 に設定される．このブルームフィルタに要素を追加するには，まず，d 個のハッシュ関数 $h_1, h_2, ..., h_d$ を利用して，d 個のアドレス $a_1, a_2, ..., a_d$ を得る．そして，それぞれのアドレスに対応するビットを 1 にする．ある要素がブルームフィルタに含まれるか判定する為には，まず，d 個のハッシュ関数から d 個のアドレス $a'_1, a'_2, ..., a'_d$ を得る．そして，それぞれのアドレスに対応するビットが 1 であるかを確認する．ここで，全てのビットが 1 だった場合は，誤りの確率を除けばその要素は含まれる．一方で，0 であるビットが含まれる場合は，その要素は含まれない．

4. 提案方式

ブルームフィルタ に文字単位でキーワードを登録することで，SEFKS を構成する．

4.1 定　　義

本方式では以下のように定義される共通鍵暗号アルゴリズム Π と，鍵付きハッシュ関数 $h_i(sk, x)$ を利用する．

$\Pi = \big(Setup(1^\lambda), Enc(sk, \cdot), Dec(sk, \cdot)\big)$ を鍵を sk とする共通鍵暗号アルゴリズムと定義する．これには，AES 暗号[6] などが挙げられる．

$Setup(1^\lambda)$　セキュリティパラメータ 1^λ を入力として，鍵 sk を出力する．

$Enc(sk, \cdot)$　鍵 sk と平文を入力として，暗号文を出力する．

$Dec(sk, \cdot)$　鍵 sk と暗号文を入力として，平文を出力する．

本方式では，m ビットの ブルームフィルタ を用いるが，ハッシュ関数として，k 個の鍵付ハッシュ関数 $h_1(sk, x), ..., h_k(sk, x)$ を利用する．$h_1, h_2, ..., h_k$ はそれぞれ原像計算困難性を持たなければならない．また，それぞれのハッシュ関数は異なる出力を持たなければならない．これには，HMAC[4] のハッシュ関数として，SHA-256[7] を利用した，HMAC-SHA256 などが挙げられる．利用するハッシュ関数の出力するビット数が m より大きい場合には，$h_i(sk, x) \bmod m\,(i \in [1, k])$ とする等，切り詰めて利用する．

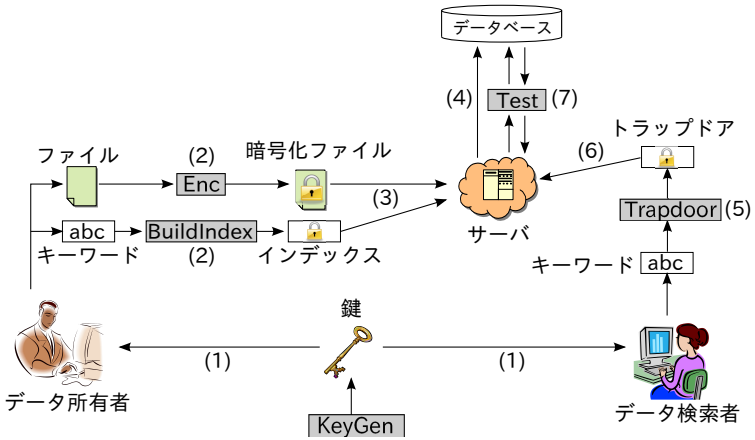

図 2　SEFKS における情報の流れ
Fig. 2　Information Flow of SEFKS

ここで，文字列を w とする時，バイト単位の文字列長を $\|w\|$ とし，i 文字目の文字を $w[i]$ と表すものとする．但し，$1 \le i \le \|w\|$ である．また，2 つの文字列 w_1, w_2 の結合を $w_1 \parallel w_2$ のように表すものとする．

提案方式では，データ所有者が登録したキーワードを w_0 とすると，検索を行うユーザは w_0 が満たすべき条件を，$w_0[i_1] = c_1 \vee (w_0[i_2] = c_2 \wedge w_0[i_3] = c_3)$ のように，積和標準形の述語論理式を指定して，検索を行うことが出来る．これを検索式と呼ぶものとする．

また，ファイル識別子を暗号化を行う対象のファイルを表す一意の識別子とする．ファイル識別子は，サーバ上の絶対パスや，一意の数値などが考えられるが，ファイルを一意に識別出来ればよく，ここではその方法について言及しない．

4.2 構　　成

本方式は以下の 4 つのアルゴリズムから構成される．

$KeyGen(1^\lambda)$　セキュリティパラメータ 1^λ を入力として，秘密鍵 sk を出力する．

$Trapdoor(sk, f)$　秘密鍵 sk と検索式 f を入力として，トラップドア $T = T_1, ..., T_n$ を出

© 2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.20
Vol.2011-IOT-13 No.20
2011/5/13

力する．

$BuildIndex(sk, FID_w, w)$　秘密鍵 sk とファイル識別子 FID_w キーワード w を入力として，暗号文 $C = \{C_I, C_{II}\}$ を出力する．キーワードが複数ある場合は，それぞれに対して実行する．

$Test(T, C)$　トラップドア T と暗号文 C を入力として，暗号文 C が指定されたトラップドアに該当するか検査し，ヒットする場合は C_{II} を出力し，ヒットしない場合は，エラー記号 $\perp$ を出力する．

これらのアルゴリズムを用いて，ユーザとサーバは図 2 に示すように以下のような処理を行う．

(1)　$KeyGen$ を用いて生成した鍵をユーザ間で共有しておく．
(2)　データを登録したいユーザは，Enc を用いてファイルを暗号化する．また，同時にファイルの内容を示すキーワードを指定し，$BuildIndex$ を用いてインデックスを生成する．
(3)　生成した暗号化ファイルと，インデックスをサーバに送信する．
(4)　サーバはデータベースにインデックスを登録する．
(5)　データを検索したいユーザは，$Trapdoor$ を用いてキーワードのトラップドアを生成する．
(6)　生成したトラップドアをサーバに送信する．
(7)　サーバは $Test$ を用いて，データを検索し，その結果をユーザに返却する．

4.3 実現方法

これらのアルゴリズムは以下のように実現される．

$KeyGen(1^\lambda)$

Π の $Setup$ をそのまま利用し，sk を出力する．

$Trapdoor(sk, f)$

(1)　$f = p_1 \vee p_2 \vee ... \vee p_n$ とする時，$i = 1, ..., n$ に対して以下の処理を行う．
　(a)　m ビットのブルームフィルタ T_i を 0 で初期化する．
　(b)　$p_i = p_{i,1} \wedge p_{i,2} \wedge ... \wedge p_{i,m_i}$ とし，$p_{i,j}$ を $w[x_{(i,j)}] = c_{(i,j)}$ と表すとするとき，$j = 1, ..., m_i$ のそれぞれについて $x_{(i,j)} \parallel c_{(i,j)}$ をブルームフィルタ T_i に追加する．
(2)　得られた $T = \{T_1, ..., T_n\}$ を出力する．

$BuildIndex(sk, FID_w, w)$

(1)　m ビットのブルームフィルタ BF を初期化する．
(2)　$1 \leq i \leq \|w\|$ の各 i に対して，BF に $i \parallel w[i]$ を追加する．
(3)　$C = \{BF, Enc(sk, FID_w \parallel h \parallel w)\}$ を出力する．但し，h はファイルのハッシュ値である．

$Test(T, C)$

各 $T_i \in T$ に対して，T_i で 1 となっているビットが，C_I においても全て 1 であれば，C_{II} を出力する．一方で，T_i において 1 となっているビットで，C_I において 0 となっているビットが 1 つでもあればエラー記号 $\perp$ を出力する．

表 1　開発環境
Table 1　Environment

OS	Windows 7 (x64)
コンパイラ	Visual C# Ver 10.0.30319.1
ランタイム	.NET Framework Ver 4.0.30319
CPU	Intel Core 2 Duo E7400 2.8GHz
メモリ	4GB

5. 評　　価

5.1 性能評価

ブルームフィルタのビット数を m，キーワード数を n とすると，暗号文のデータ量は mn ビット に n 個の $Enc(sk, FID_w \parallel h \parallel w)$ に相当するデータ量を加えたものとなる．

一方，検索用トラップドアのデータ量はブルームフィルタが OR で区切られた論理式の数 n_ℓ 個あることから，$n_\ell m$ となる．

次に，暗号化における計算量は，全てのキーワードの総文字数の回数だけハッシュ関数の計算を行うことから，キーワードの総文字数を ℓ_s とすると，暗号化における計算量は，$O(\ell_s)$ となる．

一方，検索用トラップドアの生成の計算量は，同様に論理式の数を n_ℓ とすると，$O(n_\ell)$ となる．

5.2 実装評価

部分的に類似するキーワードを含むものを検索する検索を例に取って，実装を行った．

実装を行うのに用いた開発環境は表 1 のとおりである．

共通鍵暗号に AES を用いており，更に原像計算困難性を持つ鍵付ハッシュ関数とし

© 2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.20
Vol.2011-IOT-13 No.20
2011/5/13

て HMAC-SHA256 を用いて実装を行った．HMAC-SHA256 を $h(sk, x)$ とするとき，$h_1(sk, x) = h(sk, 1 \parallel x), h_2(sk, x) = h(sk, 2 \parallel x), ..., h_n(sk, n \parallel x)$ とすることで，異なる出力を持つ鍵付ハッシュ関数として構成した．また，ブルームフィルタのビット数は 256bit とした．

用いるキーワードの文字数を変化させた時の録時や検索時に要する処理時間を図 3，図 4 に示す．キーワードの文字数が増加するに伴い，処理時間が指数関数的に増加している．この為，文字列長が大きくなる全文検索には向いていないことが分かる．しかし，本方式では暗号文全体を登録するのではなくキーワードを登録することを想定しており，1 つのキーワードが 100 文字を大きく超えるとは考えづらく，実運用上は問題とならないと考えられる．例えば，20 文字のキーワードに対するインデックス生成に要した時間は，キーワード集合生成に要した時間を含めておよそ 1.1 秒程度であった．一方，検索処理のサーバ側でインデックスとトラップドアの 1 つの組を検査するのに要する時間は 0.01 ミリ秒であった．平均 5.7 文字の合計 3,001 個のキーワードを含むデータベースに対して，“情報処理学会”というキーワードを検索すると，286 ミリ秒の時間を要した．1 つのファイルにつき平均して 3 つのキーワードを持つと仮定すると，これは約 1,000 個のファイルに相当する．評価に使用したデータベースは，SQLite と呼ばれるデータベース管理システムを用いて生成されており，インデックスを 64bit 毎に分割して格納している．インデックスとトラップドアをそれぞれ 64bit 毎に分割し，$Idx_1, Idx_2, ..., Idx_n, Td_1, Td_2, ..., Td_n$ とするとき，クエリを $Idx_1 \& Td_1 = Td_1 \wedge ... \wedge Idx_n \& Td_n = Td_n$ と構成した．今回は単純に $Test()$ を全てのキーワードに対してサーバ側で行っているが，インデックスの並び替えなどを行い，全てのキーワードの比較を必要としないようにデータ構造を最適化すれば，更に高速化が図れると考えている．このように，本提案方式においては処理に要する時間が実用的な範囲にあることが分かる．

5.3 安全性評価

SEFKS に対する攻撃としては，インデックス，トラップドアのいずれかが盗取される場合，データベースに対して改ざんが行われる場合などが考えられる．

5.3.1 インデックス，トラップドアの盗取に対する耐性

クラウド サーバの管理者がキーワードを不正に読み取ろうとした場合や，インデックスやトラップドアを送信する際に通信を悪意のある第三者に傍受され，攻撃者がキーワードの盗取を試みる場合などが該当する．

インデックス，トラップドアのいずれを盗取されたとしても構造的に大きな違いは無いため，区別する必要はない．

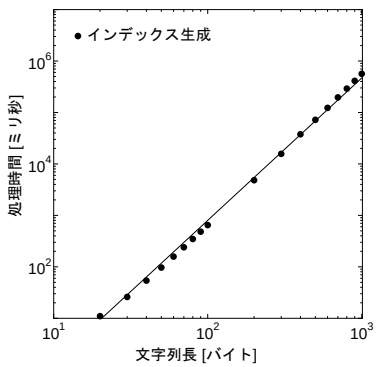

図 3 登録時の時間
Fig. 3 Processing Time of Registration

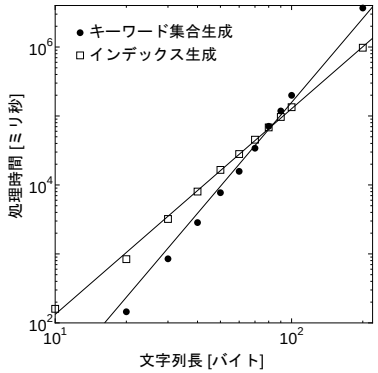

図 4 検索時の時間
Fig. 4 Processing Time of Search

得られたブルームフィルタからキーワードを復元する為に，攻撃者はまず，ブルームフィルタからハッシュ値を求め，更にハッシュ値から更にキーワードを復元する必要がある．

まず，ブルームフィルタからハッシュ値を求めることについて述べる．

ハッシュ関数は，衝突を回避するのに必要なビット数を確保する必要がある．例えば，HMAC-SHA256 を用いると 256 ビットのハッシュ値が得られる．$2^{256} \approx 1.16 \times 10^{77}$ ビットのブルームフィルタを確保することは現実的に不可能であり，偽陽性の発生確率から考えてもそのようなビット長のブルームフィルタを確保する必要もない．よって，ハッシュ値をブルームフィルタのビット数の範囲内にマッピングすることになる．その為，元のハッシュ値の情報は失われ，元のハッシュ値を予想することは困難になる．

ここで，仮に攻撃者が元のハッシュ値を得ることができたとする．その場合についても，用いるハッシュ関数の原像計算困難性により，元のキーワードは守られる．

5.3.2 データベースの改ざんに対する耐性

次に，データベースに改ざんが行われる場合について考える．ここで，データベースに対する改ざんとは，攻撃者が表 2 から表 3 に変更する場合などが該当する．この例では，2 人の預金残高が入れ換えられている．

暗号化されたデータであったとしても，サーバ管理者やサーバを乗っ取った攻撃者は，データを改ざんすることは可能であり，これを防ぐことは出来ない．特に問題となるのは，整合

© 2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.20
Vol.2011-IOT-13 No.20
2011/5/13

表 2 元のデータベース
Table 2 Original Database

名前	口座番号	預金残高
山田太郎	123456789	100,000 円
田中花子	987654321	10,000,000 円

表 3 改ざん後のデータベース
Table 3 Altered Database

名前	口座番号	預金残高
山田太郎	123456789	10,000,000 円
田中花子	987654321	100,000 円

性の取れる別の暗号文に書き換えられ，ユーザがこれを検知することができない場合である．例えば，銀行が暗号化されたデータベースで顧客の預金を管理しているとし，攻撃者が任意の顧客の預金残高を改変しようとする場合を考える．ここで，預金残高を改ざんすることでデータベース全体の整合性が取れなくなり，復号化できなくなるのであれば，銀行側はこれを検知することができ，何らかの手段で復元を試みることができる．しかし，書き換えられた状態を検知できない場合，そのまま銀行は運用を続けてしまうことになる．

そこで，本提案方式を任意のデータベースを用いて利用する場合に，改ざんを検知することが出来るかについて考察する．ここで，ファイル識別子としてファイル名を用いているものと仮定する．

ファイル識別子と暗号化ファイルの対応関係が改ざんされた場合

まず，ファイル識別子と暗号化ファイルの対応関係が改ざんされた場合を考える．例えば，file1.txt，file2.txt が存在する場合，それぞれのファイルの中身を入れ替える，というような攻撃である．この場合は，復号化時にファイルのハッシュ値がサーバから返却されたデータに含まれるハッシュ値と一致するかを確認することで検証可能である．

キーワードが書き換えられた場合

次に，キーワードが書き換えられた場合について考える．例えば，file1.txt に “キーワード 1”，file2.txt に “キーワード 2” を指定してサーバに登録している場合に，キーワードのインデックスのビットを書き換えたり，それぞれのキーワードを入れ替えたりする攻撃が挙げられる．この場合は，サーバに保存されている元のキーワードから，サーバに登録されたインデックスを構成できるか確認することで検証できる．

6. 終わりに

本研究では，データ所有者がインデックス生成時に考えうる全てのキーワードを登録する事無く，柔軟な検索を可能とする共通鍵暗号方式の検索可能暗号を提案した．

更に，提案方式を実装することで，速度，データ量のどちらも実用的な範囲に収まっていることを示した．

本提案方式では，検索式中における論理積については，データを増加させることなく実現できる．しかし，論理和を実現する場合には論理和の数だけデータ量が増加してしまう．また，処理時間も同様に増加する．今後は論理和を効率的に実現する方法についても検討していきたい．また，本方式が IND-CKA 安全などの安全性のモデルに関して安全であるかについても今後検討したい．

謝辞 本研究に際して，有益な様々なコメントを頂いた九州大学システム情報科学研究院情報学部門の堀 良彰准教授に，謹んで感謝の意を評する．

参 考 文 献

1) Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors, *Commun. ACM*, Vol.13, pp.422–426 (1970).
2) Boneh, D., DiCrescenzo, G., Ostrovsky, R. and Persiano, G.: Public Key Encryption with Keyword Search, *Advances in Cryptology - EUROCRYPT 2004* (Cachin, C. and Camenisch, J., eds.), Lecture Notes in Computer Science, Vol.3027, Springer Berlin / Heidelberg, pp.506–522 (2004). 10.1007/978-3-540-24676-3_30.
3) Knuth, D.E.: *The Art of Computer Programming*, Vol.3, Addison-Wesley Professional (1973).
4) Krawczyk, H., Bellare, M. and Canetti, R.: HMAC: Keyed-Hashing for Message Authentication, RFC 2104 (Informational) (1997).
5) Li, J., Wang, Q., Wang, C., Cao, N., Ren, K. and Lou, W.: Fuzzy Keyword Search over Encrypted Data in Cloud Computing, *INFOCOM, 2010 Proceedings IEEE*, pp.1 –5 (2010).
6) NIST: Announcing the ADVANCED ENCRYPTION STANDARDS (AES), Federal Information Processing Standards Publication 197 (2001).
7) NIST: Announcing the SECURE HASH STANDARD, Federal Information Processing Standards Publication 180-2 (2002).
8) Song, D.X., Wagner, D. and Perrig, A.: Practical techniques for searches on encrypted data, *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, pp.44 –55 (2000).
9) NPO 法人日本ネットワークセキュリティ協会セキュリティ被害調査ワーキンググループ：情報セキュリティインシデントに関する調査報告書第 1.1 版 (2011 年 1 月 27 日閲覧)，`http://www.jnsa.org/result/incident/data/2009incident_survey_v1.1.pdf` (2010).
10) 吉田麗生，小田 哲，小林鉄太郎：部分一致検索可能暗号，2010 年暗号と情報セキュリティシンポジウム (2010).

© 2011 Information Processing Society of Japan