# (強) フォワード安全な動的検索可能暗号の効率的な構成

渡邉 洋平[1,a)]　大原 一真[2,3]　岩本 貢[3]　太田 和夫[3]

**概要**：動的検索可能暗号 (Dynamic Searchable Symmetric Encryption: Dynamic SSE) は，暗号化したままの検索に加え，暗号化データベースの動的な更新も可能とする暗号技術である．Zhang ら (USENIX Security 2016) によって，フォワード安全性を満たしていない Dynamic SSE に対する実用的な攻撃が示されたことから，今やフォワード安全性は Dynamic SSE における標準的な安全性要件である．本稿ではまず，現在最も効率的なフォワード安全 Dynamic SSE として知られる Etemad ら (PoPETs 2018) の方式の証明の誤りを指摘し，その誤りを修正すると同時により効率的かつフォワード安全な方式を提案する．次に，渡邉ら (SCIS 2019) が導入した強フォワード安全性が実用上重要な安全性要件であることに言及したうえで，提案方式が強フォワード安全性を満たすように拡張できることも示す．最後に実装結果を示し，提案方式の実用性も併せて示す．

# Efficient Dynamic Searchable Encryption Schemes with (Strong) Forward Privacy

Yohei Watanabe[1,a)]　Kazuma Ohara[2,3]　Mitsugu Iwamoto[3]　Kazuo Ohta[3]

***Abstract:*** Dynamic searchable symmetric encryption (dynamic SSE) enables ones to search an encrypted database for keywords and dynamically update the database. Zhang et al. (USENIX Security 2016) showed a practical attack against dynamic SSE without forward privacy, and therefore, the security notion has become a standard security requirement for dynamic SSE. In this paper, we point out a bug in a security proof of Etemad et al.'s scheme (PoPETs 2018), which is the most efficient forward-private dynamic SSE scheme, and show how to fix it by our more efficient forward-private scheme. Next, we mention that strong forward privacy, which was introduced by Watanabe et al. (SCIS 2019), is important in practice and extend our scheme to get an efficient scheme with strong forward privacy. Finally, we provide our experimental result to show the practical performance of our schemes.

## 1. はじめに

**動的検索可能暗号** (Dynamic Searchable Symmetric Encryption: Dynamic SSE) [7] は，オンラインストレージ等の外部サーバに保存された暗号化データに対して，暗号化したまま検索を可能とし，かつ暗号化データベースも動的に更新可能であるような暗号技術である．その利便性から，これまで Dynamic SSE に関する研究が盛んに進められてきた (例えば [2], [3], [5], [7], [8], [10])．

Stefanov ら [10] は，Dynamic SSE が満たすべき安全性として，**フォワード安全性**を導入した．これは，“新たなファイルが追加される際に，そのファイルが過去に検索されたキーワードを含んでいたとしても，その事実が漏れない” という安全性である．実際，フォワード安全性を満たさない Dynamic SSE に対してファイル挿入攻撃 [12] が有効であることが知られており，フォワード安全性は Dynamic SSE における本質的な安全性要件だといえる．フォワード安全性は Bost [2] によって定式化され，これまでに多くのフォワード安全な方式 (例えば [2], [5], [8], [10]) が提案されている．現在最も効率的なフォワード安全 Dynamic SSE 方式が Etemad らの方式 [5] である (便宜上 EKPE 方式と呼ぶ)．また，渡邉ら [11] はより強い安全性要件である強フォワード安全性を導入している．

**本稿の貢献．** 本稿では，EKPE 方式の安全性証明に誤りがあることを指摘し，検索時に “取るに足らない” と考えられる情報の漏洩を許すことによって証明を修正できることを示す．誤りを修正すると同時に，EKPE 方式を改良し，これまでで最も効率的な Dynamic SSE 方式を提案する．更に，強フォワード安全性が必要だと考えられる (フォワード安全性では不十分な) 実応用先が存在することを指摘し，強フォワード安全性の実用的意義を示すと共に，ダミーエントリをうまく用いることで，提案方式が強フォワード安全性を満たすよう拡張できることを示す．最後に提案方式等の実装評価を行い，その実用性を示す．

[1] 国立研究開発法人情報通信研究機構, NICT
[2] 日本電気株式会社, NEC
[3] 電気通信大学, The University of Electro-Communications
[a)] yohei.watanabe@nict.go.jp

## 2. 準備

**記法.** 任意の自然数 $n \in \mathbb{N}$ に対して，$\{1, \ldots, n\}$ を $[n]$ と書く．有限集合 $\mathcal{X}$ に対して，$\mathcal{X}$ から一様ランダムに $x$ を取り出すことを $x \xleftarrow{\$} \mathcal{X}$，$x$ を $\mathcal{X}$ に追加することを $\mathcal{X} \leftarrow x$ と書き，$|\mathcal{X}|$ は $\mathcal{X}$ の要素数を表す．$\|$ は要素同士の連結を表す．アルゴリズムを記述する際，文字列，集合，配列は全て空に初期化されているものとする．論文を通し，$\kappa$ をセキュリティパラメータとする．任意の非対話アルゴリズム $\mathsf{A}$ に対し，$\mathsf{out} \leftarrow \mathsf{A}(\mathsf{in})$ は $\mathsf{A}$ が $\mathsf{in}$ を入力に取り $\mathsf{out}$ を出力することを表す．本稿では，クライアントとサーバ間の対話アルゴリズムを考え，$\langle \mathsf{out}_\mathrm{C}, \mathsf{out}_\mathrm{S} \rangle \leftarrow \langle \mathsf{A}_\mathrm{C}(\mathsf{in}_\mathrm{C}), \mathsf{A}_\mathrm{S}(\mathsf{in}_\mathrm{S}) \rangle$ は次を意味する：クライアントが $\mathsf{in}_\mathrm{C}$ を入力して $\mathsf{A}_\mathrm{C}$ を，サーバが $\mathsf{in}_\mathrm{S}$ を入力して $\mathsf{A}_\mathrm{S}$ をそれぞれ実行し，それぞれ $\mathsf{out}_\mathrm{C}$ と $\mathsf{out}_\mathrm{S}$ を出力として得る．簡単のために，上記を $(\mathsf{out}_\mathrm{C}; \mathsf{out}_\mathrm{S}) \leftarrow \mathsf{A}\,(\mathsf{in}_\mathrm{C}; \mathsf{in}_\mathrm{S})$ と書く．必要に応じてトランスクリプト $\mathsf{trans}$ を $\langle (\mathsf{out}_\mathrm{C}; \mathsf{out}_\mathrm{S})\,, \mathsf{trans} \rangle \leftarrow \mathsf{A}\,(\mathsf{in}_\mathrm{C}; \mathsf{in}_\mathrm{S})$ のように書く．関数 $\mathsf{negl}(\cdot)$ が任意の多項式 $\mathsf{poly}(\cdot)$ に対してある定数 $\kappa_0 \in \mathbb{N}$ が存在して，全ての $\kappa \geq \kappa_0$ に対して $\mathsf{negl}(\kappa) < 1/\mathsf{poly}(\kappa)$ が成り立つ時，$\mathsf{negl}(\cdot)$ は無視可能関数であるという．

**疑似ランダム関数.** $\pi := \{\pi_k : \{0,1\}^* \to \{0,1\}^m\}_{k \in \{0,1\}^\kappa}$ を関数族とする ($m$ は $\kappa$ の多項式).

定義 1 (PRF). 任意の確率的多項式時間アルゴリズム $\mathsf{D}$ に対し，次を満たす無視可能関数 $\mathsf{negl}(\kappa)$ が存在する時，$\pi$ を (可変長入力) 疑似ランダム関数 (Pseudo-Random Function: PRF) 族という：$|\Pr[\mathsf{D}^{\pi_k(\cdot)}(\kappa) = 1 \mid k \xleftarrow{\$} \{0,1\}^\kappa] - \Pr[\mathsf{D}^{g(\cdot)}(\kappa) = 1 \mid g \xleftarrow{\$} \mathcal{G}]| < \mathsf{negl}(\kappa)$. ただし，$\mathcal{G}$ は任意長の文字列を $m$ ビットの文字列へと写像する関数全ての集合とする．

## 3. 動的検索可能暗号

### 3.1 SSE 用の記法

$\lambda$ を $\kappa$ の多項式とし，$\Lambda := \{0,1\}^\lambda$ を取り得るキーワードの集合 (辞書と呼ぶこともある) とし[*1]，$\mathcal{F}$ を取り得る文書ファイルの集合とする．

既存研究と同様に，各文書ファイル $f_\mathsf{id} \in \mathcal{F}$ はそれぞれ内容とは無関係な識別子 (文書番号) $\mathsf{id} \in \{0,1\}^\ell$ を有し ($\ell$ は $\kappa$ の多項式)，各文書ファイル $f_\mathsf{id}$ は識別子 $\mathsf{id}$ とファイル中の異なるキーワードの集合 $\mathcal{W}_\mathsf{id} \subset \Lambda$ からなるものとする．本稿では，時系列の整理にカウンタ $t$ を用いる．$t$ は 1 に初期化され，更新・検索の度にインクリメントされる．$t$ におけるデータベース $\mathsf{DB}^{(t)}$ は，文書ファイルの集合，すなわち $\mathsf{DB}^{(t)} := \{(\mathsf{id}_i, \mathcal{W}_i)\}_{i=1}^{n^{(t)}}$ で表される ($n^{(t)}$ は $t$ の時点でサーバに保存されている文書ファイルの数を表す)．$\mathsf{DB}^{(t)}$ に含まれるキーワードの集合を $\mathcal{W}^{(t)} := \bigcup_{i=1}^{n^{(t)}} \mathcal{W}_i$ で表し，またキーワード数を $d^{(t)} := |\mathcal{W}^{(t)}|$ とする．$\mathsf{DB}^{(t)}$ のサイズ $N^{(t)}$ は $\mathsf{DB}^{(t)}$ 内の文書ファイル/キーワードペアの数で定義される (すなわち $N^{(t)} := \sum_{i=1}^{n^{(t)}} |\mathcal{W}_i|$)．$\mathsf{ID}^{(t)}$ を $\mathsf{DB}^{(t)}$ 内の識別子の集合とする (すなわち $\mathsf{ID}^{(t)} := \{\mathsf{id} \mid (\mathsf{id}, \mathcal{W}_\mathsf{id}) \in \mathsf{DB}^{(t)}\}$)．任意のキーワード $w \in \Lambda$ に対し，$\mathsf{ID}_w^{(t)}$ は $\mathsf{DB}^{(t)}$ 内の $w$ を含む文書ファイルの識別子の集合とする (すなわち $\mathsf{ID}_w^{(t)} := \{\mathsf{id} \mid \mathsf{id} \in \mathsf{ID}^{(t)} \wedge w \in \mathcal{W}_\mathsf{id}\}$)．

[*1] 衝突耐性を持つハッシュ関数を用いることで，$\Lambda := \{0,1\}^*$ とすることもできる．

### 3.2 モデル

Dynamic SSE のモデル [2], [5], [6] を示す．多くの既存研究同様，PCPA 安全な共通鍵暗号を用いることで文書ファイルの暗号化及び復号は簡単に実現可能であるため，それらは省略する (PCPA 安全性の定義は [4] 等を参照されたい)．また本研究では簡単のため，一度の追加・削除の実行に対して追加・削除される文書ファイルの数は一つとする．

まずクライアントは $\mathsf{Setup}$ アルゴリズムを実行し，秘密鍵 $k$，初期ステート情報 $\sigma^{(0)}$，初期暗号化データベース $\mathsf{EDB}^{(0)}$ を得て，$\mathsf{EDB}^{(0)}$ をサーバに送る．$t$ の時点でファイルを追加または削除する場合，クライアントとサーバは対話アルゴリズム $\mathsf{Update} = (\mathsf{Update}_\mathrm{C}, \mathsf{Update}_\mathrm{S})$ を実行する．具体的には，クライアントは $\mathsf{Update}_\mathrm{C}$ に $k$，$\sigma^{(t)}$，ラベル $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$，ラベルに対応する入力 $\mathsf{in} \in \mathsf{IN}_\mathsf{op}$ ($\mathsf{IN}_\mathsf{op}$ はセットアップ時に決まり，例えば $\mathsf{IN}_\mathsf{add} = \mathcal{F}$，$\mathsf{IN}_\mathsf{del} = \mathsf{ID}^{(t)}$) を入力し，ステート情報を更新，$\sigma^{(t+1)}$ を得る．サーバは $\mathsf{Update}_\mathrm{S}$ に $\mathsf{EDB}^{(t)}$ を入力し，更新された暗号化データベース $\mathsf{EDB}^{(t+1)}$ を得る．$t$ の時点でキーワード $q \in \Lambda$ を検索する場合，クライアントとサーバは対話アルゴリズム $\mathsf{Search} = (\mathsf{Search}_\mathrm{C}, \mathsf{Search}_\mathrm{S})$ を実行する．クライアントは $\mathsf{Search}_\mathrm{C}$ に $k$，$\sigma^{(t)}$，$q$ を入力し，更新されたステート情報 $\sigma^{(t+1)}$ 及び検索結果 $\mathcal{X}_q^{(t)}$ を得る．サーバは $\mathsf{Search}_\mathrm{S}$ に $\mathsf{EDB}^{(t)}$ を入力し，暗号化データベースを更新，$\mathsf{EDB}^{(t+1)}$ を得る．

定義 2 (SSE). 辞書 $\Lambda$ に対する Dynamic SSE $\Sigma$ は次の 3 つのアルゴリズム $\Sigma := (\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$ からなる．

- $(k, \sigma^{(0)}, \mathsf{EDB}^{(0)}) \leftarrow \mathsf{Setup}(\kappa)$: $\kappa$ を入力に取り，秘密鍵 $k$，初期ステート情報 $\sigma^{(0)}$ 初期暗号化データベース $\mathsf{EDB}^{(0)}$ を出力する非対話確率的アルゴリズム．
- $(\sigma^{(t+1)}; \mathsf{EDB}^{(t+1)}) \leftarrow \mathsf{Update}(k, \mathsf{op}, \mathsf{in}, \sigma^{(t)}; \mathsf{EDB}^{(t)})$: $\mathsf{Update}_\mathrm{C}$ 及び $\mathsf{Update}_\mathrm{S}$ からなる対話アルゴリズム．$\mathsf{Update}_\mathrm{C}$ は $k$，ラベル $\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$，ラベルに対応する入力 $\mathsf{in} \in \mathsf{IN}_\mathsf{op}$，$\sigma^{(t)}$ を入力に取り，更新されたステート情報 $\sigma^{(t+1)}$ を出力する．$\mathsf{Update}_\mathrm{S}$ は $\mathsf{EDB}^{(t)}$ を入力に取り，更新された暗号化データベース $\mathsf{EDB}^{(t+1)}$ を出力する．
- $(\sigma^{(t+1)}, \mathcal{X}_q^{(t)}; \mathsf{EDB}^{(t+1)}) \leftarrow \mathsf{Search}(k, q, \sigma^{(t)}; \mathsf{EDB}^{(t)})$: $\mathsf{Search}_\mathrm{C}$ 及び $\mathsf{Search}_\mathrm{S}$ からなる対話アルゴリズム．$\mathsf{Search}_\mathrm{C}$ は $k$，検索キーワード $q \in \Lambda$，$\sigma^{(t)}$ を入力に取り，更新されたステート情報 $\sigma^{(t+1)}$ 及び検索結果 $\mathcal{X}_q^{(t)}$ を出力する．$\mathsf{Search}_\mathrm{S}$ は $\mathsf{EDB}^{(t)}$ を入力に取り，更新された暗号化データベース $\mathsf{EDB}^{(t+1)}$ を出力する．

紙面の都合上，$\Sigma$ の正当性については省略する．厳密な正当性の定義は [3] を参照されたい．

**Real Experiment:** $\mathsf{Real}_{\mathsf{D}}(\kappa, Q)$

$(k, \sigma^{(0)}, \mathsf{EDB}^{(0)}) \leftarrow \mathsf{Setup}(\kappa)$
$\mathsf{st}_{\mathsf{D}} := \{\mathsf{EDB}^{(0)}\}$
**for** $t = 1$ **to** $Q$ **do**
  $\mathsf{query} \leftarrow \mathsf{D}_t(\mathsf{st}_{\mathsf{D}})$
  **if** $\mathsf{query} = (\mathsf{upd}, \mathsf{op}, \mathsf{in})$ **then**
    $\langle(\sigma^{(t)}; \mathsf{EDB}^{(t)}), \mathsf{trans}^{(t)}\rangle \leftarrow \mathsf{Update}(k, \mathsf{op}, \mathsf{in}, \sigma^{(t-1)}; \mathsf{EDB}^{(t-1)})$
  **if** $\mathsf{query} = (\mathtt{srch}, q)$ **then**
    $\langle(\sigma^{(t)}, \mathcal{X}_q^{(t-1)}; \mathsf{EDB}^{(t)}), \mathsf{trans}^{(t)}\rangle \leftarrow \mathsf{Search}(k, q, \sigma^{(t-1)}; \mathsf{EDB}^{(t-1)})$
  $\mathsf{st}_{\mathsf{D}} \leftarrow (\mathsf{EDB}^{(t)}, \mathsf{trans}^{(t)})$
$b \leftarrow \mathsf{D}_{Q+1}(\mathsf{st}_{\mathsf{D}})$
**return** $b$

**Ideal Experiment:** $\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q)$

$(\mathsf{EDB}^{(0)}, \mathsf{st}_{\mathsf{S}}) \leftarrow \mathsf{S}_0(\mathcal{L}_{\mathsf{Setup}}(\kappa))$
$\mathsf{st}_{\mathsf{D}} := \{\mathsf{EDB}^{(0)}\}$
**for** $t = 1$ **to** $Q$ **do**
  $\mathsf{query} \leftarrow \mathsf{D}_t(\mathsf{st}_{\mathsf{D}})$
  **if** $\mathsf{query} = (\mathsf{upd}, \mathsf{op}, \mathsf{in})$ **then**
    $\langle(\mathsf{st}'_{\mathsf{S}}; \mathsf{EDB}^{(t)}), \mathsf{trans}^{(t)}\rangle \leftarrow \mathsf{S}_t(\mathsf{st}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Upd}}(t, \mathsf{op}, \mathsf{in}); \mathsf{EDB}^{(t-1)})$
  **if** $\mathsf{query} = (\mathtt{srch}, q)$ **then**
    $\langle(\mathsf{st}'_{\mathsf{S}}; \mathsf{EDB}^{(t)}), \mathsf{trans}^{(t)}\rangle \leftarrow \mathsf{S}_t(\mathsf{st}_{\mathsf{S}}, \mathcal{L}_{\mathsf{Srch}}(t, q); \mathsf{EDB}^{(t-1)})$
  $\mathsf{st}_{\mathsf{D}} \leftarrow (\mathsf{EDB}^{(t)}, \mathsf{trans}^{(t)})$
  $\mathsf{st}_{\mathsf{S}} := \mathsf{st}'_{\mathsf{S}}$
$b \leftarrow \mathsf{D}_{Q+1}(\mathsf{st}_{\mathsf{D}})$
**return** $b$

図 1　試行 $\mathsf{Real}_{\mathsf{D}}$ 及び試行 $\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}$.

### 3.3 $\mathcal{L}$-適応的安全性

多くの既存研究同様，シミュレーションベースで安全性を定義する．(Dynamic) SSE では，安全性レベルと効率性のトレードオフがあり，効率的な検索のためにはいくらかの情報漏洩を許容する必要があることが知られている．そのような情報漏洩は**漏洩関数** $\mathcal{L} := (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Upd}}, \mathcal{L}_{\mathsf{Srch}})$ として特徴づけられる．直感的には，$\mathcal{L}_{\mathsf{Setup}}$，$\mathcal{L}_{\mathsf{Upd}}$，$\mathcal{L}_{\mathsf{Srch}}$ は，それぞれセットアップ時，更新時，検索時に漏洩する情報を表す．$\mathcal{L}$-適応的安全性とは，そのような漏洩関数 $\mathcal{L}$ で示される情報漏洩を許したうえでそれ以上の情報は一切漏らさないという，Dynamic SSE における標準的な安全性である．具体的には，確率的多項式時間アルゴリズム $\mathsf{D} = (\mathsf{D}_1 \ldots, \mathsf{D}_{Q+1})$ とクライアント間の試行 $\mathsf{Real}_{\mathsf{D}}$ と D とシミュレータ $\mathsf{S} = (\mathsf{S}_0, \ldots, \mathsf{S}_Q)$ 間の試行 $\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}$ の 2 つで定義される (各試行は図 1 参照).

定義 3 ($\mathcal{L}$-適応的安全性). $\Sigma$ を Dynamic SSE 方式とする．任意の確率的多項式時間アルゴリズム D に対し，$|\Pr[\mathsf{Real}_{\mathsf{D}}(\kappa, Q) = 1] - \Pr[\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q) = 1]| < \mathsf{negl}(\kappa)$ を満たす確率的多項式時間アルゴリズム S が存在するならば，$\Sigma$ は $\mathcal{L}$-適応的安全であるという．

### 3.4 フォワード安全性

一言で言えば，**フォワード安全性** [2] とは，“新たに追加されたファイルが過去の検索キーワードを含むかどうかの情報を漏らさない” ことを保証する．Zhang ら [12] はファイル挿入攻撃というフォワード安全でない Dynamic SSE に対する攻撃を提案した．ファイル挿入攻撃は実際に起こり得る攻撃であることから，現在ではフォワード安全性を満たすことが Dynamic SSE の最低限の安全性要件となっている．

定義 4 (フォワード安全性 [2]). $\Sigma$ を $\mathcal{L}$-適応的安全な Dynamic SSE 方式とする ($\mathcal{L} = (\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Upd}}, \mathcal{L}_{\mathsf{Srch}})$)．$\mathcal{L}_{\mathsf{Upd}}$ (特に $\mathsf{op} = \mathsf{add}$ の時) を $\mathcal{L}_{\mathsf{Upd}}(t, \mathsf{add}, \mathsf{in}) = \mathcal{L}'(t, \mathsf{add}, (\mathsf{id}, |\mathcal{W}_{\mathsf{id}}|, |f_{\mathsf{id}}|))$ と表すことのできる時，$\Sigma$ はフォワード安全であるという．ただし，in は文書ファイル $f_{\mathsf{id}}$ に対応する入力であり，$\mathcal{L}'$ はステートレスな関数である．

## 4. 提案構成法

本節では，まず EKPE 方式 [5] の安全性証明に誤りがあることに言及し，その修正方法を示す．更に，単に安全性証明を修正するだけではなく，EKPE 方式を改良し，よりシンプルかつ効率的なフォワード安全 Dynamic SSE 方式を提案する．

### 4.1 EKPE 方式

EKPE 方式は，既存のフォワード安全性を満たす方式の中で最も効率的な Dynamic SSE 方式として知られている．

$\pi = \{\pi_k : \{0,1\}^* \to \{0,1\}^\eta\}_{k \in \{0,1\}^\kappa}$ 及び $h = \{h_k : \{0,1\}^* \to \{0,1\}^\kappa\}_{k \in \{0,1\}^\kappa}$ を可変長入力 PRF 族とする ($\eta$ は $\kappa$ の多項式)．EKPE 方式では，各キーワード $q \in \mathcal{W}^{(t)}$ ごとに ($q \in \Lambda$ ではないことに注意)，ファイルカウンタ $\mathsf{fc}_q$ と検索カウンタ $\mathsf{sc}_q$ の 2 種類のカウンタをステート情報として用いる．$\mathsf{fc}_q$ はキーワード $q$ を含むファイルがサーバに何個保存されているかを表し，$q$ に対応するアドレスをこのカウンタを基に $\pi$ を用いて生成することで，検索時に生成するトラップドアの個数を必要最低限に抑え，効率的な検索を実現している．ただし，フォワード安全性を達成するためにこれらの “$q$ に対応するアドレス” は $q$ の検索の度に更新される．具体的には，検索の度に $\pi$ の PRF 鍵を更新し，再び $\mathsf{fc}_q$ を基にアドレスを生成することでアドレスの更新を行う．PRF 鍵の更新は，これまでに $q$ が何回検索されたかを表す $\mathsf{sc}_q$ を基に $h$ を用いて PRF 鍵を生成することで実現する．$q$ の検索後，たとえ $q$ を含むファイルが新たに追加されたとしても，そのファイル分の “$q$ に対応するアドレス” は更新後の PRF 鍵で生成されるため，更新前の PRF 鍵から得られた情報 (すなわち過去の検索結果) は役に立たない．

EKPE 方式の構成を図 2 に示す．[5] では，EKPE 方式は次の漏洩関数 $\mathcal{L} := \{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Upd}}, \mathcal{L}_{\mathsf{Srch}}\}$ に関して $\mathcal{L}$-適応的安全かつフォワード安全であると述べられている：

$$\mathcal{L}_{\mathsf{Setup}}(\kappa) = \Lambda, \quad \mathcal{L}_{\mathsf{Upd}}(t, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}})) = (\mathsf{id}, |\mathcal{W}_{\mathsf{id}}|, |f_{\mathsf{id}}|),$$
$$\mathcal{L}_{\mathsf{Upd}}(t, \mathsf{del}, \mathsf{id}) = (\mathsf{id}, |\mathcal{W}_{\mathsf{id}}|), \quad \mathcal{L}_{\mathsf{Srch}}(t, q) = (\mathsf{SP}_q^{(t)}, \mathsf{AP}_q^{(t)}).$$

**EKPE 方式:** $\mathsf{Search}(k, q, \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

$k_q^{(\mathsf{sc}_q)} := h(k, 1\|q\|\mathsf{sc}_q)$
**for** $i = 1$ **to** $\mathsf{fc}_q$ **do**
  $\mathtt{addr}_{q,i} := \pi(k_q^{(\mathsf{sc}_q)}, 0\|i)$
  $\mathtt{mask}_{q,i} := \pi(k_q^{(\mathsf{sc}_q)}, 1\|i)$
  $\mathcal{T}_q^{(t)} \leftarrow (\mathtt{addr}_{q,i}, \mathtt{mask}_{q,i})$ // 検索トラップドアとして追加
Send $\mathsf{trans}_1^{(t)} := \mathcal{T}_q^{(t)}$ to the server

**Server:**

**for** $\forall(\mathtt{addr}_{q,i}, \mathtt{val}_{q,i}) \in \mathcal{T}_q^{(t)}$ **do**
  **if** $\mathsf{IndexW}[\mathtt{addr}_{q,i}] \neq \mathrm{NULL}$ **then**
    $\mathcal{X}_q^{(t)} \leftarrow \mathsf{IndexW}[\mathtt{addr}_{q,i}] \oplus \mathtt{mask}_{q,i}$ // i.e., $\mathcal{X}_q^{(t)} \leftarrow \mathsf{id}_i$
    $\mathsf{IndexW}[\mathtt{addr}_{q,i}] := \mathrm{NULL}$ // 古いアドレスを削除
    Find $\mathtt{addr}_{q,i}^{(f)}$ s.t. $\mathsf{IndexF}[\mathtt{addr}_{q,i}^{(f)}] = \mathtt{addr}_{q,i}$
    $\mathsf{IndexF}[\mathtt{addr}_{q,i}^{(f)}] := \mathrm{NULL}$ // IndexF の対応箇所も削除
Send $\mathsf{trans}_2^{(t)} := \mathcal{X}_q^{(t)}$ to the client

**Client:**

$\mathsf{sc}_q := \mathsf{sc}_q + 1$
$\mathsf{fc}_q := |\mathcal{X}_q^{(t)}|$ // 前回検索以降に削除されたファイルの分を圧縮
$\hat{k}_q^{(\mathsf{sc}_q)} := h(k, 1\|q\|\mathsf{sc}_q)$
**for** $j = 1$ **to** $\mathsf{fc}_q$ **do**
  $\widehat{\mathtt{addr}}_{q,j} := \pi(\hat{k}_q^{(\mathsf{sc}_q)}, 0\|j)$ // 次回検索用のアドレスを生成
  $\widehat{\mathtt{val}}_{q,j} := \mathsf{id}_j \oplus \pi(\hat{k}_q^{(\mathsf{sc}_q)}, 1\|j)$ // $\mathsf{id}_j \in \mathcal{X}_q^{(t)}$
  $\widehat{\mathcal{A}}_q^{(t)} \leftarrow (\widehat{\mathtt{addr}}_{q,j}, \widehat{\mathtt{val}}_{q,j})$
Send $\mathsf{trans}_3^{(t)} := \widehat{\mathcal{A}}_q^{(t)}$ to the server
$\sigma^{(t+1)} := \big((\mathsf{sc}_w, \mathsf{fc}_w)_{w \in \mathcal{W}^{(t+1)}}, \{\mathsf{cnt}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{ID}^{(t+1)}}\big)$
**return** $(\sigma^{(t+1)}, \mathcal{X}_q^{(t)})$

**Server:**

**for** $\forall(\widehat{\mathtt{addr}}_{q,j}, \widehat{\mathtt{val}}_{q,j}) \in \widehat{\mathcal{A}}_q^{(t)}$ **do**
  $\mathsf{IndexW}[\widehat{\mathtt{addr}}_{q,j}] := \widehat{\mathtt{val}}_{q,j}$
  $\mathsf{IndexF}[\mathtt{addr}_{q,j}^{(f)}] := \widehat{\mathtt{addr}}_{q,j}$ // 12 行目で空にした所に格納
**return** $\mathsf{EDB}^{(t+1)} := (\mathsf{IndexF}, \mathsf{IndexW})$

**EKPE 方式:** $\mathsf{Update}(k, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}}), \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

$\mathsf{cnt}_{\mathsf{id}} := |\mathcal{W}_{\mathsf{id}}|$ // $\mathcal{W}_{\mathsf{id}} = \{w_i\}_{i=1}^{\mathsf{cnt}_{\mathsf{id}}}$
$k_{\mathsf{id}} := h(k, 0\|\mathsf{id})$ // IndexF 用の PRF 鍵を生成
**for** $i = 1$ **to** $\mathsf{cnt}_{\mathsf{id}}$ **do**
  **if** $\mathsf{fc}_{w_i}$ is undefined **then**
    $\mathsf{fc}_{w_i} := 0$
  **if** $\mathsf{sc}_{w_i}$ is undefined **then**
    $\mathsf{sc}_{w_i} := 0$
  $\mathsf{fc}_{w_i} := \mathsf{fc}_{w_i} + 1$
  $k_{w_i}^{(\mathsf{sc}_{w_i})} := h(k, 1\|w_i\|\mathsf{sc}_{w_i})$ // IndexW 用の PRF 鍵を生成
  $\mathtt{addr}_i^{(f)} := \pi(k_{\mathsf{id}}, i)$ // IndexF 用のアドレス
  $\mathtt{addr}_{w_i}^{(w)} := \pi(k_{w_i}^{(\mathsf{sc}_{w_i})}, 0\|\mathsf{fc}_{w_i})$ // IndexW 用のアドレス
  $\mathtt{val}_{w_i}^{(w)} := \mathsf{id} \oplus \pi(k_{w_i}^{(\mathsf{sc}_{w_i})}, 1\|\mathsf{fc}_{w_i})$ // $\mathtt{addr}_{w_i}^{(w)}$ の格納値
  $\mathcal{U}_{\mathsf{id}}^{(t)} \leftarrow (\mathtt{addr}_i^{(f)}, \mathtt{addr}_{w_i}^{(w)}, \mathtt{val}_{w_i}^{(w)})$
Send $\mathsf{trans}_1^{(t)} := \mathcal{U}_{\mathsf{id}}^{(t)}$ to the server
**return** $\sigma^{(t+1)} := \big((\mathsf{sc}_w, \mathsf{fc}_w)_{w \in \mathcal{W}^{(t+1)}}, \{\mathsf{cnt}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{ID}^{(t+1)}}\big)$

**Server:**

**for** $\forall(\mathtt{addr}_i^{(f)}, \mathtt{addr}_{w_i}^{(w)}, \mathtt{val}_{w_i}^{(w)}) \in \mathcal{U}_{\mathsf{id}}^{(t)}$ **do**
  $\mathsf{IndexF}[\mathtt{addr}_i^{(f)}] := \mathtt{addr}_{w_i}^{(w)}$ // 削除用インデックス
  $\mathsf{IndexW}[\mathtt{addr}_{w_i}^{(w)}] := \mathtt{val}_{w_i}^{(w)}$ // 検索用インデックス
**return** $\mathsf{EDB}^{(t+1)} := (\mathsf{IndexF}, \mathsf{IndexW})$

**EKPE 方式:** $\mathsf{Update}(k, \mathsf{del}, \mathsf{id}, \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

$k_{\mathsf{id}} := h(k, 0\|\mathsf{id})$ // IndexF 用の PRF 鍵を生成
**for** $i = 1$ **to** $\mathsf{cnt}_{\mathsf{id}}$ **do**
  $\mathtt{addr}_i^{(f)} := \pi(k_{\mathsf{id}}, i)$ // 削除するアドレスを生成
Send $\mathsf{trans}_1^{(t)} := (k_{\mathsf{id}}, \mathcal{U}_{\mathsf{id}}^{(t)} := \{\mathtt{addr}_i^{(f)}\}_{i \in [\mathsf{cnt}_{\mathsf{id}}]})$ to the server
**return** $\sigma^{(t+1)} := \big((\mathsf{sc}_w, \mathsf{fc}_w)_{w \in \mathcal{W}^{(t+1)}}, \{\mathsf{cnt}_{\mathsf{id}}\}_{\mathsf{id} \in \mathsf{ID}^{(t+1)}}\big)$

**Server:**

**for** $\forall \mathtt{addr}_i^{(f)} \in \mathcal{U}_{id}^{(t)}$ **do**
  $\mathsf{IndexW}[\mathsf{IndexF}[\mathtt{addr}_i^{(f)}]] := \mathrm{NULL}$
  $\mathsf{IndexF}[\mathtt{addr}_i^{(f)}] := \mathrm{NULL}$
**return** $\mathsf{EDB}^{(t+1)} := (\mathsf{IndexF}, \mathsf{IndexW})$

**図 2** EKPE 方式．Setup は $k \xleftarrow{\$} \{0,1\}^\kappa$，$\sigma^{(0)} := \emptyset$，$\mathsf{EDB}^{(0)} := (\mathsf{IndexF}[], \mathsf{IndexW}[])$ を出力する ($\mathsf{IndexF}[], \mathsf{IndexW}[]$ は空の配列)．

ここで，$\mathsf{SP}_q^{(t)}$ は検索パターン ($q$ の検索履歴) と呼ばれ，過去に $q$ が検索された時のカウンタの集合 ($t$ を含む) で表され，$\mathsf{AP}_q^{(t)}$ は $q$ の検索結果 (すなわち $\mathsf{AP}_q^{(t)} = \mathcal{X}_q^{(t)}$) のことであり，アクセスパターンと呼ばれる．しかしながら，その証明には，検索時のシミュレートに関する次のような誤りがある．検索クエリ $(t, q)$ に対し，シミュレートすべきトランスクリプトは $\mathsf{trans}_1^{(t)} := \mathcal{T}_q^{(t)}, \mathsf{trans}_2^{(t)} := \mathcal{X}_q^{(t)}, \mathsf{trans}_3^{(t)} := \widehat{\mathcal{A}}_q^{(t)}$ である．この時，$|\mathcal{T}_q^{(t)}| = \mathsf{fc}_q$ であり，すなわちシミュレータ S は $\mathsf{fc}_q$ 個の検索トラップドアをランダムに選ぶ必要がある．ここで，$\mathsf{fc}_q$ は追加によって増えるが，削除によって減らないカウンタである．$\mathsf{Real}_{\mathsf{D}}(\kappa, Q)$ では追加時に文書ファイルが見えるため $\mathsf{fc}_q$ を正しく管理できる一方，$\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q)$ では $\mathcal{L}$ のみを用いて追加・削除・検索をシミュレートしなければならないため，$\mathsf{fc}_q$ の把握は困難である．特に，前回の検索クエリ $(t', q)$ から今回の検索クエリ $(t, q)$ の間に，$q$ を含むあるファイルの追加・削除が行われた場合を考える．$\mathsf{fc}_q$ を正しく把握可能な $\mathsf{Real}_{\mathsf{D}}(\kappa, Q)$ では問題なく $\mathsf{fc}_q$ 個のトラップドアを $\mathsf{trans}_1^{(t)}$ として計算可能である．しかし，$\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q)$ では，$\mathsf{fc}_q$ $(= |\mathcal{X}_q^{(t')}| + 1)$ であるにも関わらず，漏洩情報として S が把握しているのは前回の検索結果 $\mathcal{X}_q^{(t')}$ と今回の検索結果 $\mathcal{X}_q^{(t)}$ $(|\mathcal{X}_q^{(t')}| = |\mathcal{X}_q^{(t)}|)$ であるため，正しい数のトラップドアをシミュレートすることができない．

証明を修正するためには，検索時に “前回の検索クエリ $(t', q)$ から今回の検索クエリ $(t, q)$ の間に追加及び削除がなされた $q$ を含むファイルの識別子” ($\mathsf{TempID}_q^{(t)} := (\bigcup_{j=t'}^{t-1} \mathsf{ID}_q^{(j)}) \setminus \mathsf{ID}_q^{(t)}$ とする) の漏洩も許す必要がある．$\mathsf{TempID}_q^{(t)}$ から新たに得られる情報は，上記の “検索クエリ間に追加・削除されたファイル $f_{\mathsf{id}}$ が $q$ を含んでいた” という情報であり (当然 $f_{\mathsf{id}}$ や $q$ は漏れていない)，特にフォワード安全性には影響がないと考えられる．次節で $\mathsf{TempID}_q^{(t)}$ の漏洩を許すことで正しく安全性を証明できることを示す．

$\Sigma$: Search$(k, q, \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

$k_q^{(\mathsf{sc}_q)} := h(k, q \| \mathsf{sc}_q)$
**for** $i = 1$ **to** $\mathsf{fc}_q$ **do**
  $\mathcal{T}_q^{(t)} \leftarrow \pi(k_q^{(\mathsf{sc}_q)}, i)$ // 検索トラップドアを生成
Send $\mathsf{trans}_1^{(t)} := \mathcal{T}_q^{(t)}$ to the server

**Server:**

**for** $\forall \mathtt{addr} \in \mathcal{T}_q^{(t)}$ **do**
  **if** $\mathsf{Index}[\mathtt{addr}] \neq \mathrm{NULL}$ **then**
    $\mathcal{X}_q^{(t)} \leftarrow \mathsf{Index}[\mathtt{addr}]$
    $\mathsf{Index}[\mathtt{addr}] := \mathrm{NULL}$ // 古いアドレスを削除
Send $\mathsf{trans}_2^{(t)} := \mathcal{X}_q^{(t)}$ to the client

**Client:**

$\mathsf{sc}_q := \mathsf{sc}_q + 1$
$\mathsf{fc}_q := |\mathcal{X}_q^{(t)}|$ // 前回検索以降に削除されたファイルの分を圧縮
$\hat{k}_q^{(\mathsf{sc}_q)} := h(k, q \| \mathsf{sc}_q)$
**for** $j = 1$ **to** $\mathsf{fc}_q$ **do**
  $\widehat{\mathcal{A}}_q^{(t)} \leftarrow \pi(\hat{k}_q^{(\mathsf{sc}_q)}, j)$ // 次回検索用のアドレスを生成
Send $\mathsf{trans}_3^{(t)} := \widehat{\mathcal{A}}_q^{(t)}$ to the server
$\sigma^{(t+1)} := \{(\mathsf{sc}_w, \mathsf{fc}_w)\}_{w \in \mathcal{W}^{(t+1)}}$
**return** $(\sigma^{(t+1)}, \mathcal{X}_q^{(t)})$

**Server:**

**for** $j = 1$ **to** $|\widehat{\mathcal{A}}_q^{(t)}|$ **do**
  $\mathsf{Index}[\widehat{\mathtt{addr}}_j] := \mathsf{id}_j$ // $\widehat{\mathcal{A}}_q^{(t)} = \{\widehat{\mathtt{addr}}_j\}_{j=1}^{\mathsf{fc}_q}$, $\mathcal{X}_q^{(t)} = \{\mathsf{id}_j\}_{j=1}^{\mathsf{fc}_q}$
**return** $\mathsf{EDB}^{(t+1)} := \mathsf{Index}$

$\Sigma$: Update$(k, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}}), \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

**for** $\forall w \in \mathcal{W}_{\mathsf{id}}$ **do**
  **if** $\mathsf{fc}_w$ is undefined **then**
    $\mathsf{fc}_w := 0$
  **if** $\mathsf{sc}_w$ is undefined **then**
    $\mathsf{sc}_w := 0$
  $\mathsf{fc}_w := \mathsf{fc}_w + 1$
  $k_w^{(\mathsf{sc}_w)} := h(k, w \| \mathsf{sc}_w)$ // PRF 鍵を生成
  $\mathcal{U}_{\mathsf{id}}^{(t)} \leftarrow \pi(k_w^{(\mathsf{sc}_w)}, \mathsf{fc}_w)$ // $q$ 用のアドレスを新たに追加
Send $\mathsf{trans}_1^{(t)} := (\mathsf{id}, \mathcal{U}_{\mathsf{id}}^{(t)})$ to the server
**return** $\sigma^{(t+1)} := ((\mathsf{sc}_w, \mathsf{fc}_w)_{w \in \mathcal{W}^{(t+1)}})$

**Server:**

**for** $\forall \mathtt{addr} \in \mathcal{U}_{\mathsf{id}}^{(t)}$ **do**
  $\mathsf{Index}[\mathtt{addr}] := \mathsf{id}$
**return** $\mathsf{EDB}^{(t+1)} := \mathsf{Index}$

$\Sigma$: Update$(k, \mathsf{del}, \mathsf{id}, \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

Send $\mathsf{trans}_1^{(t)} := \mathsf{id}$ to the server
**return** $\sigma^{(t+1)} := \sigma^{(t)}$

**Server:**

**for** $\forall \mathtt{addr} \in \mathcal{A}_{\mathsf{id}}$ **do** // $\mathcal{A}_{\mathsf{id}} := \{\mathtt{addr} \mid \mathsf{Index}[\mathtt{addr}] = \mathsf{id}\}$
  $\mathsf{Index}[\mathtt{addr}] := \mathrm{NULL}$
**return** $\mathsf{EDB}^{(t+1)} := \mathsf{Index}$

図 3　提案方式 $\Sigma = (\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$．Setup は EKPE 方式とほぼ同様なため省略．

### 4.2　フォワード安全な構成法

本節では，EKPE 方式の正しい安全性証明を与えるだけではなく，より効率的かつシンプルな Dynamic SSE 方式 $\Sigma = (\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$ を示す (図 3 参照)．効率性の比較は 6 節で行う．

定理 1. $\pi$ 及び $h$ が可変長入力 PRF 族であれば，図 3 のように構成した Dynamic SSE $\Sigma = (\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$ は，任意の $t$ 及び任意の $q \in \Lambda$ に対し，以下が成り立つような漏洩関数 $\mathcal{L} := \{\mathcal{L}_{\mathsf{Setup}}, \mathcal{L}_{\mathsf{Upd}}, \mathcal{L}_{\mathsf{Srch}}\}$ に関して $\mathcal{L}$-適応的安全かつフォワード安全である：

$$\mathcal{L}_{\mathsf{Setup}}(\kappa) = \Lambda, \quad \mathcal{L}_{\mathsf{Upd}}(t, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}})) = (\mathsf{id}, |\mathcal{W}_{\mathsf{id}}|, |f_{\mathsf{id}}|),$$
$$\mathcal{L}_{\mathsf{Upd}}(t, \mathsf{del}, \mathsf{id}) = \mathsf{id}, \quad \mathcal{L}_{\mathsf{Srch}}(t, q) = (\mathsf{SP}_q^{(t)}, \mathsf{AP}_q^{(t)}, \mathsf{TempID}_q^{(t)}).$$

ただし，$\mathsf{TempID}_q^{(t)} := (\bigcup_{j=t'}^{t-1} \mathsf{ID}_q^{(j)}) \setminus \mathsf{AP}_q^{(t)}$ であり，$t'$ は最後に $q$ が検索された時のカウンタである ($t$ で初めて $q$ が検索された場合は $t' = 1$ とする)．

証明．　まず，各 PRF ($h(k, \cdot)$ 及び $\pi(k_w^{(\mathsf{sc}_w)}, \cdot)$) が真正ランダム関数に置き換わった試行 $\mathsf{Real}_{\mathsf{D}}(\kappa, Q)$ を $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ とする．また，試行 $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ 中で任意の $t \in [Q]$ に対して $\mathsf{EDB}^{(t)}$ 中の任意の 2 つのアドレスが衝突する事象を Col とする．紙面の都合上詳細は省略するが，$\mathsf{Real}_{\mathsf{D}}(\kappa, Q)$ と $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ の差は可変長入力 PRF 族の安全性から無視でき，また Col が起きる確率は真正ランダム関数の性質から無視できる．従って本証明では，任意の確率的多項式時間アルゴリズム D に対し，$|\Pr[\mathsf{Real}'_{\mathsf{D}}(\kappa, Q) = 1 \wedge \neg\mathsf{Col}] - \Pr[\mathsf{Sim}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q) = 1]| < \mathsf{negl}(\kappa)$ が成り立つことを示す．

具体的には，どのように $\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q)$ におけるシミュレータ S を構成するかを示し，D が S から得る情報と Col が起きない $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ における同情報とを情報理論的に識別できないことを示す．また本証明中では，一度削除されたファイルが再登録される際には識別子は新しいものに変わるものとする．仮に同じ識別子を用いるとサーバに同じファイルが登録されたという事実が漏れることから，このような運用が実用的にも望ましい．以下では，D のクエリに対する Update 及び Search のシミュレーションを示す．

query $= (\mathsf{upd}, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}}))$ の場合：試行 $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ では，Update$(k, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}}), \sigma^{(t)}; \mathsf{EDB}^{(t)})$ が実行され，対応するトランスクリプト $\mathsf{trans}^{(1)}$ は id 及び $|\mathcal{W}_{\mathsf{id}}|$ 個の $\eta$ ビットの乱数列 $\mathcal{U}_{\mathsf{id}}^{(t)}$ である．この乱数列は $\mathsf{EDB}^{(t)}$ 中の id を格納するアドレスとなり，また $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ ではこれらアドレスの衝突は起きないことに留意されたい．従って，$\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q)$ において，S は $\eta$ ビットの文字列を $|\mathcal{W}_{\mathsf{id}}|$ 個ランダムに選び，$\mathcal{U}_{\mathsf{id}}^{(t)}$ とすればよい．ただし，この時選ぶ文字列は過去に一度でもアドレスとして利用されたものは除外し，未使用のものを選ぶ．

query $= (\mathsf{upd}, \mathsf{del}, \mathsf{id})$ の場合：試行 $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ では，クライアントは $\mathsf{trans}^{(1)} := \mathsf{id}$ を送り，サーバは id を含んでいるアドレスを全て NULL に設定する．$\mathcal{L}_{\mathsf{Upd}}(t, \mathsf{del}, \mathsf{id}) = \mathsf{id}$ であるから，S はこれを容易にシミュレート可能である．

query $= (\mathtt{srch}, q)$ の場合：試行 $\mathsf{Real}'_{\mathsf{D}}(\kappa, Q)$ では，まずクラ

イアントが各 $i \in [\mathsf{fc}_q]$ に対して $\mathtt{addr}^{(i)}_{q,\mathsf{sc}_q} := g_{q,\mathsf{sc}_q}(i)$ を計算し，サーバに $\mathsf{trans}^{(1)} := \mathcal{T}^{(t)}_q = \{\mathtt{addr}^{(1)}_{q,\mathsf{sc}_q}, \ldots, \mathtt{addr}^{(\mathsf{fc}_q)}_{q,\mathsf{sc}_q}\}$ をトラップドアとして送る ($g_{q,\mathsf{sc}_q}$ は $(q, \mathsf{sc}_q)$ に対応する真正ランダム関数)．ここで，Col は起きないことから，検索結果は常に $\mathcal{X}^{(t)}_q = \mathsf{ID}^{(t)}_q$ が成り立つ．サーバは，$\mathcal{T}^{(t)}_q$ に含まれる各アドレスが格納している識別子全て (すなわち検索結果 $\mathcal{X}^{(t)}_q$) を $\mathsf{trans}^{(2)}$ として送り，古いアドレスの格納値は削除する．この時，前回の $q$ の検索後に $q$ を含むファイルが削除されている可能性があるため，$\mathsf{fc}_q \geq |\mathcal{X}^{(t)}_q|$ であり，必ずしも $\mathsf{fc}_q = |\mathcal{X}^{(t)}_q|$ とは限らない[*2]．正当性より，$\mathcal{X}^{(t)}_q$ に含まれる各識別子が $\mathcal{T}^{(t)}_q$ に含まれる $|\mathcal{X}^{(t)}_q|$ 個のアドレスそれぞれに格納されており，それ以外の $\mathsf{fc}_q - |\mathcal{X}^{(t)}_q|$ 個のアドレスは空である．ただし，それらの空アドレスは元々 $\mathsf{TempID}^{(t)}_q$ の各識別子が格納されていたものであり，シミュレートする際にはその点にも留意する必要がある．$\mathsf{trans}^{(2)}$ を受信した後，クライアントは $\mathsf{sc}_q := \mathsf{sc}_q + 1$ 及び $\mathsf{fc}_q := |\mathcal{X}^{(t)}_q|$ とし，新たな真正ランダム関数 $g_{q,\mathsf{sc}_q}$ を選ぶ．各 $i \in [\mathsf{fc}_q]$ に対して新たなアドレス $\widehat{\mathtt{addr}}^{(i)}_{q,\mathsf{sc}_q} := g_{q,\mathsf{sc}_q}(i)$ を計算し，$\mathsf{trans}^{(3)} := \widehat{\mathcal{A}}^{(t)}_q = \{\widehat{\mathtt{addr}}^{(1)}_{q,\mathsf{sc}_q}, \ldots, \widehat{\mathtt{addr}}^{(\mathsf{fc}_q)}_{q,\mathsf{sc}_q}\}$ をサーバに送る．サーバは $\mathcal{X}^{(t)}_q$ の各識別子を $\widehat{\mathcal{A}}^{(t)}_q$ の各アドレスに格納する．

上記を基に，試行 $\mathsf{Ideal}_{\mathsf{D},\mathsf{S},\mathcal{L}}(\kappa, Q)$ における $\mathsf{S}$ を $\mathcal{L}_{\mathsf{Srch}}(t, q) = (\mathsf{SP}^{(t)}_q, \mathsf{AP}^{(t)}_q, \mathsf{TempID}^{(t)}_q)$ を用いて次のように構成する．まず、$\mathsf{trans}^{(t)}_1 := \mathcal{T}^{(t)}_q$ をシミュレートするにあたり，次の 2 通りを考える必要がある．

(a) $q$ が初めて検索された場合，すなわち $\mathsf{SP}^{(t)}_q = \{t\}$．各 $\mathsf{id} \in (\mathcal{X}^{(t)}_q \cup \mathsf{TempID}^{(t)}_q)$ に対し，$\mathcal{U}^{(t')}_{\mathsf{id}}$ からいまだトラップドアとして使用されていない乱数列を選ぶ ($t'$ は id が追加された時のカウンタである)．

(b) $q$ が以前にも検索されている場合，すなわち $\mathsf{SP}^{(t)}_q \neq \{t\}$．前回の検索時のカウンタを $t' \in \mathsf{SP}^{(t)}_q$ とする．検索結果 $\mathcal{X}^{(t)}_q$ のうち，前回の検索結果 $\mathcal{X}^{(t')}_q$ にも含まれていた識別子 $(\mathsf{id} \in (\mathcal{X}^{(t)}_q \cap \mathcal{X}^{(t')}_q))$ はその際に再登録したアドレスをトラップドアとして割り当て，そうでない識別子 $(\mathsf{id} \in ((\mathcal{X}^{(t)}_q \setminus \mathcal{X}^{(t')}_q) \cup \mathsf{TempID}^{(t)}_q))$ は (1) と同様にトラップドアとして割り当てれば良い．

$\mathsf{trans}^{(2)}$ は $\mathsf{AP}^{(t)}_q$ をそのまま用いれば良く，$\mathsf{trans}^{(3)} := \widehat{\mathcal{A}}^{(t)}_q$ は次のように設定すれば良い：各 $\mathsf{id} \in \mathsf{AP}^{(t)}_q$ に対し，アドレスとして未使用の $\eta$ ビットの文字列 $\widehat{\mathtt{addr}}_{q,\mathsf{id}}$ をランダムに選び，$\widehat{\mathcal{A}}^{(t)}_q$ に加えれば良い． □

## 5. 強フォワード安全性と提案方式の拡張

定義 4 から明らかなように，フォワード安全性を満たしていたとしても，攻撃者 (すなわちサーバ) は文書ファイル $f_{\mathsf{id}}$ の追加時に当該ファイルの異なるキーワード数 $|\mathcal{W}_{\mathsf{id}}|$ を知ることができる．実際，既存のフォワード安全な Dynamic SSE 方式 (例えば [2], [5], [8], [10]) 及び 4.2 節の構成法はこのような異なるキーワード数の漏洩を許しており，また許しているからこそ安全性が証明できる構成となっている．通常の共通鍵暗号であれば，攻撃者が暗号文から得られる情報はその長さ $|f_{\mathsf{id}}|$ のみであり，攻撃者から見て $2^{|f_{\mathsf{id}}|}$ 個の平文候補が存在する[*3]．一方で，Dynamic SSE では，サーバは文書ファイルの長さ $|f_{\mathsf{id}}|$ だけでなく当該ファイル中の異なるキーワード数 $|\mathcal{W}_{\mathsf{id}}|$ も知っていることから，更なる候補の絞り込みが可能であり，その意味で明らかに共通鍵暗号より安全性が弱い．ここで問題となるのは “その異なるキーワード数の漏洩が実用上どの程度の脅威となり得るか” であり，既存研究ではこれに関する議論がほとんど行われていない．しかしながら，そのような漏洩が脅威となり得る例を以下に示す．

**STR (Short Tandem Repeat) 法．** STR とは DNA サンプル中の特定の遺伝子座に現れる 2 から 7 個ほどの塩基からなる短い DNA 配列の反復数のことであり，STR は個々人によって異なるうえ，確実に両親からその反復数を遺伝することが知られている．STR 法とは，STR の性質を用いて個人の特定や親子関係の解析を行う手法であり，実際に FBI の DNA データベース CODIS で採用されている手法である．ここで，DNA データベース上での Dynamic SSE 方式の利用を考える．すなわち，各 DNA サンプルがデータベースに暗号化されて保存されており，Dynamic SSE を用いて検索を行う状況を想定する[*4]．たとえ Dynamic SSE 方式がフォワード安全性を満たしているとしても，新しい DNA サンプルが追加される度にそのサンプルが含む異なるキーワード数 (すなわち DNA 配列の種類数) が漏洩する．例えば，異なるキーワード数が比較的少ない場合，DNA サンプル中に同じ DNA 配列が繰り返し登場している可能性があることを意味し，STR に関する情報が漏れるおそれがある．従って，DNA データベースに Dynamic SSE を適用する場合は，フォワード安全性でも不十分である．

渡邉ら [11] は，異なるキーワード数すら漏れない強フォワード安全性を定義している．上記の DNA データベースのような，異なるキーワード数の漏洩が脅威となり得るアプリケーションにおいては，強フォワード安全性を満たすことが重要であるといえる．

定義 5 (強フォワード安全性 [11]). $\Sigma$ を $\mathcal{L}$-適応的安全な Dynamic SSE 方式とする．$\mathcal{L}_{\mathsf{Upd}}$ (特に $\mathsf{op} = \mathsf{add}$ の時) を $\mathcal{L}_{\mathsf{Upd}}(t, \mathsf{add}, \mathsf{in}) = \mathcal{L}'(t, \mathsf{add}, (\mathsf{id}, |f_{\mathsf{id}}|))$ と表すことのできる時，$\Sigma$ は強フォワード安全であるという．ただし，in は文書ファイル $f_{\mathsf{id}}$ に対応する入力であり，$\mathcal{L}'$ はステートレスな関数である．

強フォワード安全性を達成するための構成アプローチは，**追加の際にダミーアドレスを過不足なく追加すること**である．ここで，[4], [11] のように，$f_{\mathsf{id}}$ が含むことのできるキーワードの最大個数 $\mathsf{max}_{\mathsf{id}}$ の概念を導入する．すなわち，最も小さいキーワード $w \in \Lambda$ から順にそのサイズを足していき，$|f_{\mathsf{id}}|$ を超える手前までに足したキーワードの合計個数

[*2] この点の見落としが EKPE 方式の証明の誤りに繋がっている．

[*3] ここではあくまで候補の数についてのみ議論している．選択暗号文攻撃や既知の平文分布から更なる絞り込みが可能である．

[*4] 当該 DNA データベースの目的によって何をキーワードとするかは変わるが，ここでは短い DNA 配列パターン (STR に関連するものも含む) がキーワードとして登録されていると仮定する．

表 1　既存方式と提案方式の比較 (いずれもフォワード安全)．$\widehat{N}^{(t)}$ は $\mathsf{DB}^{(t)}$ 中の各ファイル $f_{\mathsf{id}}$ の $\mathsf{max}_{\mathsf{id}}$ の合計を表す．$\mathsf{op} \in \{\mathsf{add}, \mathsf{del}\}$ に対して，$n^{[t]}_{w,\mathsf{op}}$ はプロトコルの最初から，$n^{(\mathtt{srch})}_{w,\mathsf{op}}$ は $w$ の最後の検索から，今までに $w$ に関して $\mathsf{op}$ が行われた回数を指し，$p$ はスレッド数 (論理プロセッサ数) を表す．

| | $\lvert\sigma^{(t)}\rvert$ | $\lvert\mathsf{EDB}^{(t)}\rvert$ | 検索コスト | 更新コスト | 強フォワード安全性 |
|---|---|---|---|---|---|
| SPS14 [10] | $\mathcal{O}(\sqrt{N^{(t)}})$ | $\mathcal{O}(N^{(t)})$ | $\mathcal{O}(n^{(t)}_w \log^3 N^{(t)})$ | $\mathcal{O}(\lvert\mathcal{W}_{\mathsf{id}}\rvert \log^2 N^{(t)})$ | |
| Bos16 [2] | $\mathcal{O}(d^{(t)})$ | $\mathcal{O}(N^{(t)})$ | $\mathcal{O}(n^{[t]}_{w,\mathsf{add}} + n^{[t]}_{w,\mathsf{del}})$ | $\mathcal{O}(\lvert\mathcal{W}_{\mathsf{id}}\rvert)$ | |
| KKL$^+$17 [8] | $\mathcal{O}(d^{(t)})$ | $\mathcal{O}(N^{(t)})$ | $\mathcal{O}(n^{[t]}_{w,\mathsf{add}})$ | $\mathcal{O}(\lvert\mathcal{W}_{\mathsf{id}}\rvert)$ | |
| EKPE18 [5] | $\mathcal{O}(d^{(t)} + n^{(t)})$ | $\mathcal{O}(N^{(t)})$ | $\mathcal{O}((n^{(t)}_w + n^{(\mathtt{srch})}_{w,\mathsf{del}})/p)$ | $\mathcal{O}(\lvert\mathcal{W}_{\mathsf{id}}\rvert/p)$ | |
| WIO19 [11] | $\mathcal{O}(n^{(t)})$ | $\mathcal{O}(\widehat{N}^{(t)})$ | $\mathcal{O}(n^{(t)}/p)$ | $\mathcal{O}(\mathsf{max}_{\mathsf{id}}/p)$ | ✓ |
| 提案方式 (§4.2) | $\mathcal{O}(d^{(t)})$ | $\mathcal{O}(N^{(t)})$ | $\mathcal{O}((n^{(t)}_w + n^{(\mathtt{srch})}_{w,\mathsf{del}})/p)$ | $\mathcal{O}(\lvert\mathcal{W}_{\mathsf{id}}\rvert/p)$ | |
| 拡張 (§5) | $\mathcal{O}(d^{(t)})$ | $\mathcal{O}(\widehat{N}^{(t)})$ | $\mathcal{O}((n^{(t)}_w + n^{(\mathtt{srch})}_{w,\mathsf{del}})/p)$ | $\mathcal{O}(\mathsf{max}_{\mathsf{id}}/p)$ | ✓ |

が $\mathsf{max}_{\mathsf{id}}$ である．ファイルを追加する際に，$\mathsf{max}_{\mathsf{id}} - |\mathcal{W}_{\mathsf{id}}|$ 個のダミーアドレスを追加することで，$|\mathcal{W}_{\mathsf{id}}|$ の漏洩を抑えることができる．4.2 節で示した構成法は EKPE 方式と比べ簡素化されており，これによりダミーの挿入が容易になっている．具体的には，次のような更新手順を考える．

$\Sigma$: $\mathsf{Update}(k, \mathsf{add}, (\mathsf{id}, \mathcal{W}_{\mathsf{id}}), \sigma^{(t)}; \mathsf{EDB}^{(t)})$

**Client:**

$k_{\mathsf{id}} := h(k, 1\|\mathsf{id})$ // ダミー用の鍵を生成
**for** $\forall w \in \mathcal{W}_{\mathsf{id}}$ **do**
  **if** $\mathsf{fc}_w$ is undefined **then**
    $\mathsf{fc}_w := 0$
  **if** $\mathsf{sc}_w$ is undefined **then**
    $\mathsf{sc}_w := 0$
  $\mathsf{fc}_w := \mathsf{fc}_w + 1$
  $k_w^{(\mathsf{sc}_w)} := h(k, 0\|w\|\mathsf{sc}_w)$ // 0/1 で $k_w^{(\mathsf{sc}_w)}$ と $k_{\mathsf{id}}$ の衝突を防ぐ
  $\mathcal{U}_{\mathsf{id}}^{(t)} \leftarrow \pi(k_w^{(\mathsf{sc}_w)}, \mathsf{fc}_w)$
**for** $\beta = 1$ **to** $\mathsf{max}_{\mathsf{id}} - |\mathcal{W}_{\mathsf{id}}|$ **do**
  $\mathcal{U}_{\mathsf{id}}^{(t)} \leftarrow \pi(k_{\mathsf{id}}, \beta)$ // ダミーを生成
Send $\mathsf{trans}_1^{(t)} := (\mathsf{id}, \mathcal{U}_{\mathsf{id}}^{(t)})$ to the server
**return** $\sigma^{(t+1)} := ((\mathsf{sc}_w, \mathsf{fc}_w)_{w \in \mathcal{W}^{(t+1)}})$

**Server:**

**for** $\forall \mathtt{addr} \in \mathcal{U}_{\mathsf{id}}^{(t)}$ **do**
  $\mathsf{Index}[\mathtt{addr}] := \mathsf{id}$
**return** $\mathsf{EDB}^{(t+1)} := \mathsf{Index}$

この変更に合わせ，提案方式の Search アルゴリズムの 1 及び 12 行目の $h$ への入力を $q\|\mathsf{sc}_q$ から $0\|q\|\mathsf{sc}_q$ に修正する必要がある．

定理 2. $\pi$ 及び $h$ が可変長入力 PRF 族であれば，上記のように図 3 の構成を修正した Dynamic SSE $\Sigma = (\mathsf{Setup}, \mathsf{Update}, \mathsf{Search})$ は，任意の $t$ 及び任意の $q \in \Lambda$ に対し，定理 1 と同様の漏洩関数 $\mathcal{L}$ に関して $\mathcal{L}$-適応的安全かつ強フォワード安全である．

## 6. 効率評価および実装結果

本節では，提案方式の効率評価を行うと共に，C++を用いた提案方式の実装結果を示す．まず，既存のフォワード安全 Dynamic SSE 方式との性能比較を表 1 に示す．EKPE18 方式は EKPE 方式に本稿で示した安全性証明の修正を与えたものである．EKPE18 方式に比べ，提案方式は特にステート情報長が効率的になっている他，Index に識別子をそのまま格納する構成となっており，その意味でシンプルかつ効率的になっている (図 2 及び 3 参照)．提案方式はいずれもステート情報長が WIO19 方式に劣るものの，EKPE18 方式同様，検索コストの効率性が非常に高い．また，EKPE18 方式及び WIO19 方式同様，検索及び更新処理を並列化することが可能である．

今回，4.2 節のフォワード安全な方式 (Simplified EKPE scheme)，5 節の強フォワード安全な方式 (EKPE-based scheme)，WIO19 方式 (CGKO-based scheme) の 3 つを同環境で実装し，性能を比較した．本実験は Amazon EC2 の m4.2xlarge インスタンス上で行い，OS は Ubuntu Server 18.04 LTS (HVM, 32 GiB メモリ，8 CPU コア), ストレージは EBS General Purpose (SSD) Volume Type を選択した．PRF として，Intel AES-NI 命令セットを用いた AES-GCM 及び GMAC を利用した (鍵長は 128 ビット)．また，それらの実装には OpenSSL ライブラリ (version 1.1.0g) 中の EVP API を利用した．本実装はシングルスレッド (表 1 における $p = 1$) で行われているが，いずれの方式も並列化が可能であり，更なる効率化も期待できる．

**データセット．** 今回，データセットとして Enron Email dataset [1] (May 7, 2015 version) を用いた．このデータセットにはおよそ 150 人のユーザ分の約 2.4GB のメールデータが含まれている．表 2 に当該データセット中のファイル数，キーワード数，データセットの総サイズを示す．EDB 中のキーワードはデータセットに登場する単語の語幹部分のみとし， Porter stemming アルゴリズム [9] を用いて単語から語幹を抽出することでキーワード集合を作成した (Porter stemming アルゴリズムの実装は NLTK (Natural Language ToolKit) ライブラリを用いた)．なお，あらかじめヘッダ，記号，URL 等の情報は取り除いた．表 3 に各ファイル $f_{\mathsf{id}}$ のサイズ $|f_{\mathsf{id}}|$，$\mathsf{max}_{\mathsf{id}}$ 及び $|\mathcal{W}_{\mathsf{id}}|$ の統計を示す．

**追加・削除コスト．** 図 4 に示した通り，(通信時間も含め) 517,401 ファイル全てを追加するのにかかった時間は約 75–80 秒であり，紙面の都合上図は省略するが，全てのファ

表 2　Enron email dataset

| #files | #keywords | Size (KB) |
|---|---|---|
| 517,401 | 214,874 | 2,413,971 |

表 3　データセットの統計情報.

| parameter | max | min | average |
|---|---|---|---|
| $\lvert f_{\mathsf{id}}\rvert$ (bytes) | 2,011,957 | 398 | 4,445 |
| $\mathsf{max}_{\mathsf{id}}$ | 251,495 | 58 | 343.7 |
| $\lvert\mathcal{W}_{\mathsf{id}}\rvert$ | 59148 | 12 | 77.1 |
| $\mathsf{max}_{\mathsf{id}} - \lvert\mathcal{W}_{\mathsf{id}}\rvert$ | 192,347 | 46 | 266.6 |

イルの削除に要する時間は約 2 秒ほどであった．従って，1 ファイルあたりの追加・削除に要する時間は平均およそ 150 マイクロ秒および 3.8 マイクロ秒と見積もることができる．図 4 からわかるように，驚くことに強フォワード安全性を達成するにあたって必要なダミーアドレスの追加にかかる時間はほとんど実用性に影響を与えていない．

**検索コスト．** 図 5 及び 6 はそれぞれ，WIO19 方式及び提案方式の検索コストを示している．縦軸は一つの検索クエリにかかった所要時間 (通信コストを含む) を表し，横軸はそれぞれ，図 5 は EDB に登録されているファイルの総数を，図 6 は EDB 中の検索キーワード $q$ を含むファイル数を表す．WIO19 方式は，$q$ を含むファイルがいくつ存在するかに関わらず，検索コストが常にファイル総数に依存 ($\mathcal{O}(n^{(t)})$) する．しかしながら，200,000 個ものファイルが登録されていても検索にかかる時間は約 0.7 秒と，十分実用的である．一方で，提案方式の検索コストは $\mathcal{O}(n_w^{(t)} + n_{w,\mathsf{del}}^{(\mathtt{srch})})$ であり，非常に効率的である．実際，検索キーワードを含むファイルが 200,000 個存在する場合は，総ファイル数が何個であろうとも，約 1.6 秒で検索を実行でき，非常に大きなデータベースにも適用可能である．

**謝辞**　本研究は JSPS 科研費 JP17H01752, JP18K11293, JP18K19780, JP18H05289, JP18H03238 の助成を受けています．STR 法に関してご助言いただきましたヒューマノーム研究所 瀬々潤博士に感謝いたします．

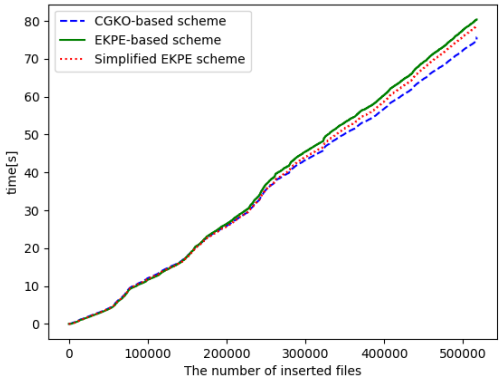

図 4　各方式の追加コスト.

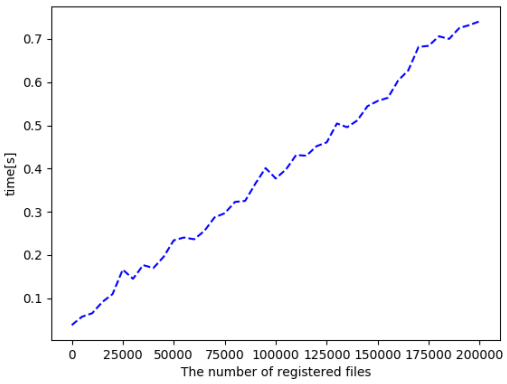

図 5　WIO19 方式 [11] の検索コスト.

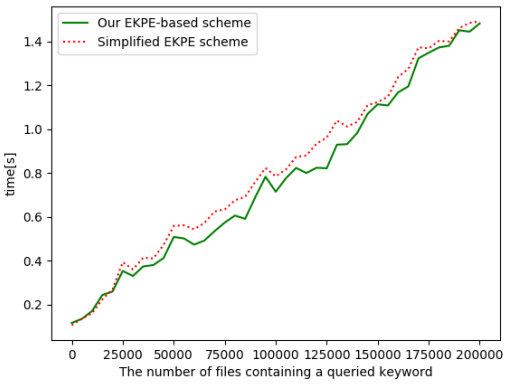

図 6　提案方式の検索コスト.

## 参考文献

[1] Enron email dataset (may 7, 2015 version), 2015.

[2] R. Bost. $\Sigma o\varphi o\varsigma$: Forward secure searchable encryption. In *ACM CCS 2016*, pages 1143–1154, 2016.

[3] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*, 2014.

[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM CCS 2006*, pages 79–88, 2006.

[5] M. Etemad, A. Küpçü, C. Papamanthou, and D. Evans. Efficient dynamic searchable encryption with forward privacy. In *Proceedings of PETs (PoPETs)*, volume 2018(1), pages 5–20, 2018.

[6] J. Ghareh Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili. New constructions for forward and backward private symmetric searchable encryption. In *ACM CCS 2018*, pages 1038–1055, 2018.

[7] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM CCS 2012*, pages 965–976, 2012.

[8] S. K. Kim, M. Kim, D. Lee, J. H. Park, and W.-H. Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In *ACM CCS 2017*, pages 1449–1463, 2017.

[9] M. Porter. The English (Porter2) Stemming Algorithm. Available at `http://snowball.tartarus.org/algorithms/english/stemmer.html`, 2001.

[10] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*, 2014.

[11] 渡邉, 岩本, 太田. 効率的でフォワード安全な動的検索可能暗号. In *SCIS 2019* **予稿集**, 3C1-3, 2019.

[12] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security 2016*, pages 707–720, 2016.