辞書なし検索可能暗号について

茨城大学　　黒澤　馨
東京工業大学　尾形わかは

2018年12月21日

SITA 2018　特別セッション
（情報セキュリティ）

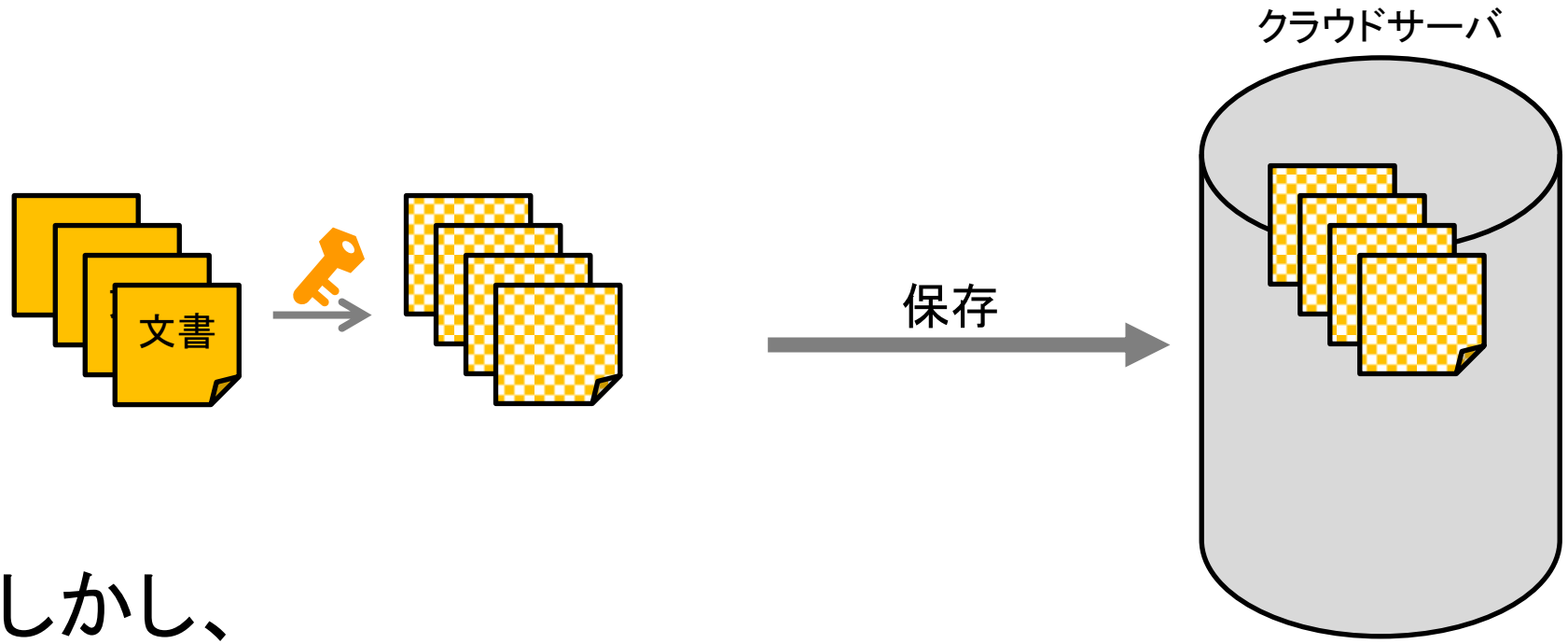
クラウド・ストレージ・サービス

- Google Drive
- OneDrive
- Amazon Drive
- …

などが商用化されている。

各文書は

- 暗号化してサーバに保存した方が安全



しかし、
　そうするとキーワード検索ができない

検索可能暗号 (SSE)

- この相反する2つの問題を解決する手段
- 共通鍵暗号に基づく暗号方式
- Searchable Symmetric Encryption (SSE)

SSEの研究

- Song, Wagner, Perrig (2000)
- 以来、多くの論文

CRYPTO 2018でも、２件の発表

- Demertzis, Papadopoulos, Papamanthou
- Asharov, Segev, Shahaf

講演のアウトライン

- SSE方式の具体例
- 辞書ありSSE方式について

  黒澤、大瀧:

  Financial Cryptography 2012
- 辞書なしSSE方式について

  尾形、黒澤:

  Financial Cryptography 2017

SSE方式の具体例

キーワード	文書1	文書2	文書3	文書4	文書5
福島	1	0	1	0	1
茨城	0	1	0	1	0
ハワイ	1	1	1	0	0

索引

クライアント

各文書を暗号化

	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
福島	(1	0	1	0	1)
茨城	(0	1	0	1	0)
ハワイ	(1	1	1	0	0)

クライアント

各キーワードを、鍵KのAES暗号で暗号化

	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
AES(K, 福島)	(1	0	1	0	1)
AES(K, 茨城)	(0	1	0	1	0)
AES(K, ハワイ)	(1	1	1	0	0)

クライアント

各キーワードを別の鍵K’で暗号化し、
各行に足す

	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
AES(K, 福島)	(1	0	1	0	1) ＋AES(K’, 福島)
AES(K, 茨城)	(0	1	0	1	0) ＋AES(K’, 茨城)
AES(K, ハワイ)	(1	1	1	0	0) ＋AES(K’, ハワイ)

クライアントは

この暗号化テーブルをサーバに送り、
秘密鍵 (K,K') をkeep

	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
AES(K, 福島)	(1	0	1	0	1) ＋AES(K', 福島)
AES(K, 茨城)	(0	1	0	1	0) ＋AES(K', 茨城)
AES(K, ハワイ)	(1	1	1	0	0) ＋AES(K', ハワイ)

サーバ

茨城 で検索したい場合

クライアントは、

AES(K, 茨城), AES(K’, 茨城)

を計算し、サーバに送る

サーバ

	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
AES(K, 福島)	(1	0	1	0	1) ＋AES(K’, 福島)
AES(K, 茨城)	(0	1	0	1	0) ＋AES(K’, 茨城)
AES(K, ハワイ)	(1	1	1	0	0) ＋AES(K’, ハワイ)

キーワード 茨城 で検索したい

クライアントは、

AES(K, 茨城), AES(K’, 茨城)

を計算し、サーバに送る

サーバ

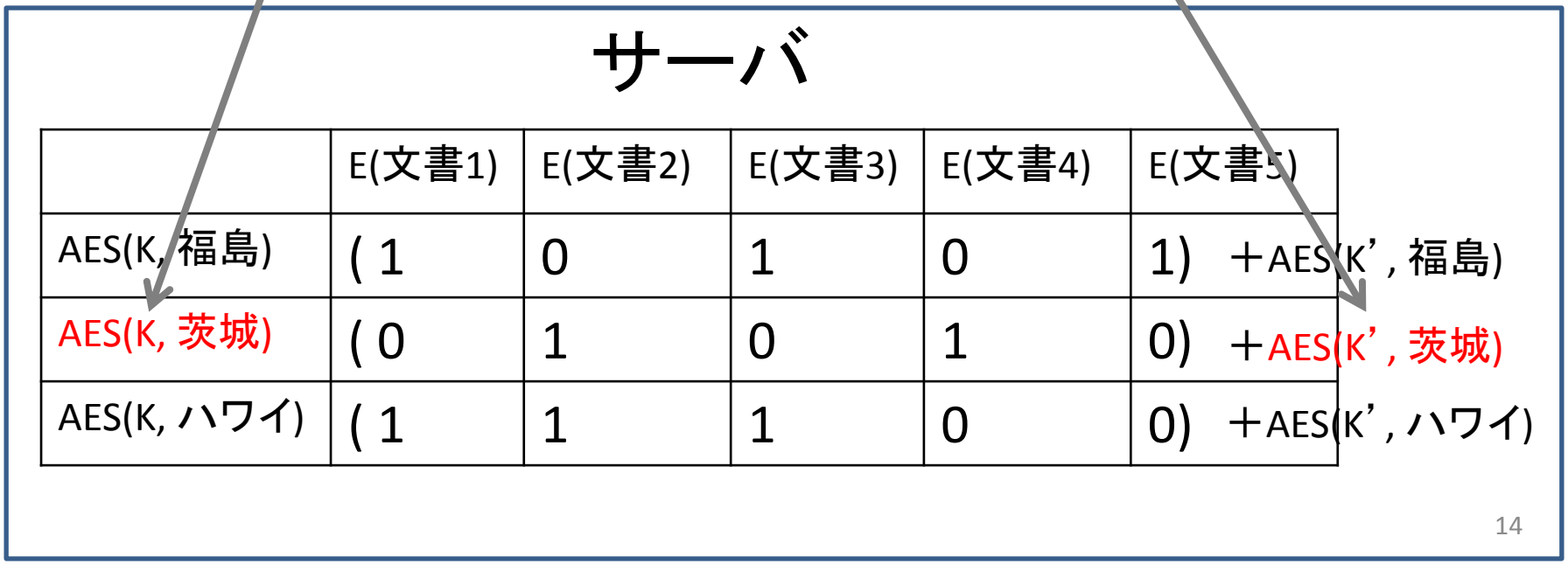
	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
AES(K, 福島)	(1	0	1	0	1) ＋AES(K’, 福島)
AES(K, 茨城)	(0	1	0	1	0) ＋AES(K’, 茨城)
AES(K, ハワイ)	(1	1	1	0	0) ＋AES(K’, ハワイ)

キーワード 茨城 で検索したい

クライアントは、

AES(K, 茨城), AES(K’, 茨城)

を計算し、サーバに送る



検索フェーズ

	E(文書1)	E(文書2)	E(文書3)	E(文書4)	E(文書5)
AES(K, 福島)	(1	0	1	0	1) ＋AES(K’, 福島)
AES(K, 茨城)	(0	1	0	1	0) ＋AES(K’, 茨城)
AES(K, ハワイ)	(1	1	1	0	0) ＋AES(K’, ハワイ)

サーバは、(0 1 0 1 0)を復号。
2番目と4番目が1なので、
E(文書2), E(文書4)をクライアントへ返す。

SSE方式の安全性 (1)

- 文書数、キーワード数、アクセスパターン等、
  「最小漏れ情報」
  はサーバにわかってもいい、と割り切る。

- それ以上の情報が漏れないとき、
  passiveな敵（サーバ）に対し安全
- Curtmola et al. (2006)が定式化

SSEの安全性 (2)

Passiveな敵（サーバ）

- 正しい検索結果を返す。
- 前述の方式は、Passiveな敵に対し安全
- もっと効率の良い方式が存在

Activeな敵（サーバ）

- 間違った検索結果を返す。

Activeな敵

辞書 ＝ キーワード集合

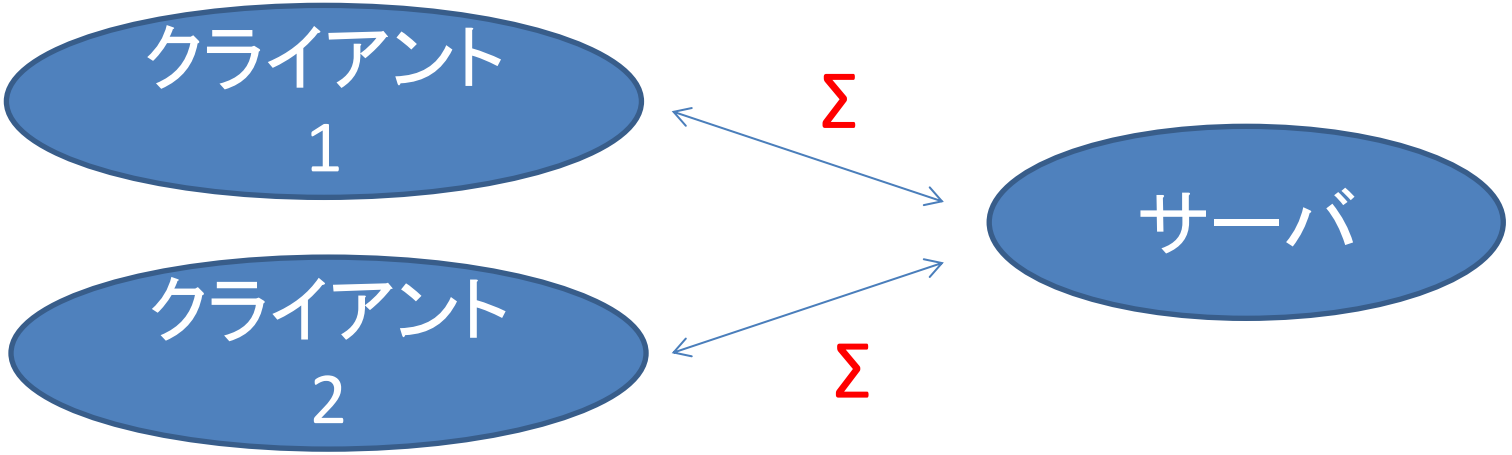
- クライアントが辞書を覚えている場合、
  Activeな敵に対し安全なSSE方式

  黒澤・大瀧 (FC 2012)が構成
  さらに、UC安全性を証明

一般に

プロトコル Σ が単体で安全であったとしても

- Σ が同時並行的に実行されたり
- Σ が大きなプロトコルに組み込まれた場合、



もはや安全とは限らない。

Universal Composability (UC)

- Σ が同時並行的に実行されたり
- Σ が大きなプロトコルに組み込まれた場合でも

Σが安全であることを保障するフレームワーク

Canetti (2001)が導入

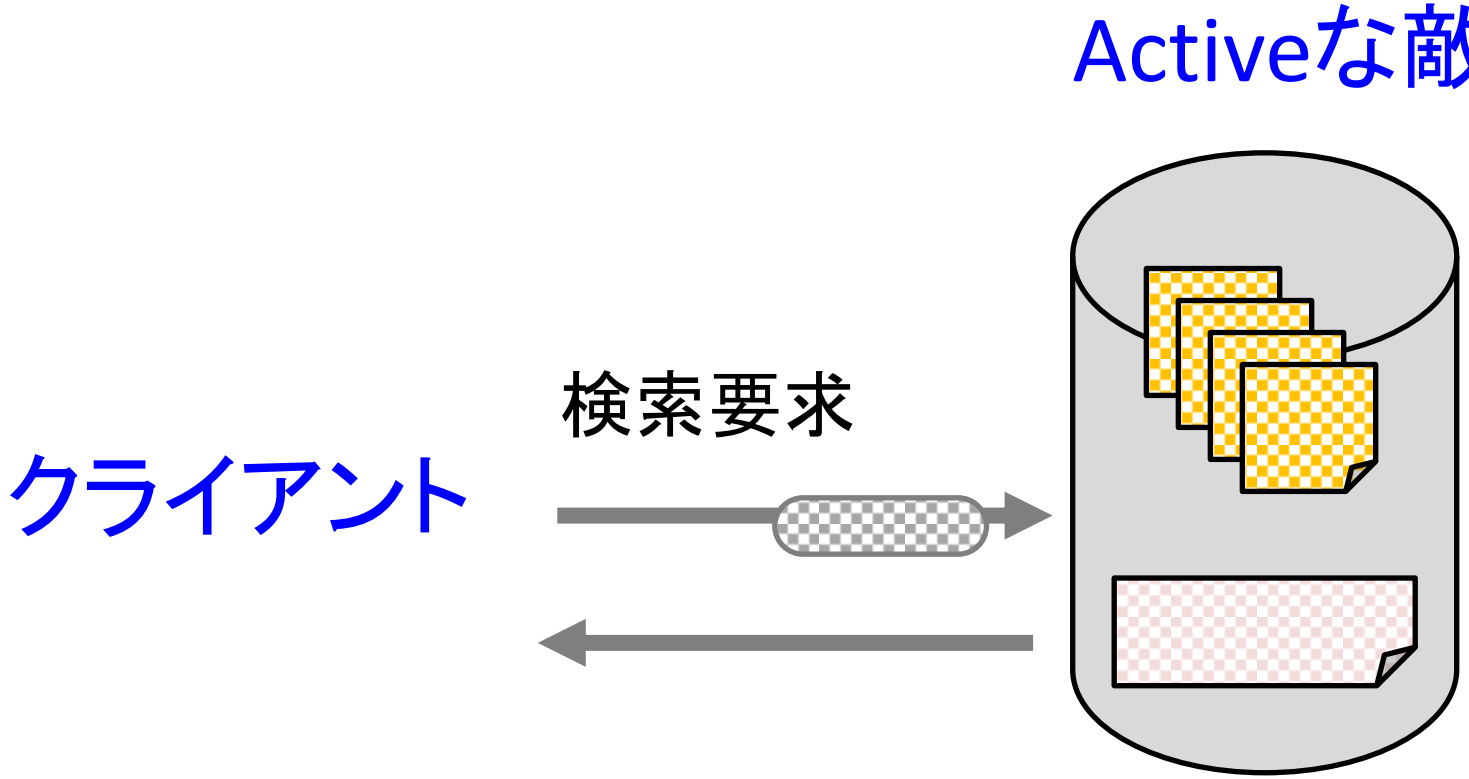
UC安全性＝とても強い安全性

黒澤、大瀧 (FC 2012)

辞書ありSSE方式について

- Activeな敵に対するプロトコル単体の安全性を定式化
- UC安全性との等価性を証明
- 提案方式のUC安全性を証明

クライアントが
辞書を覚えていない場合



ヒットする文書なし

- 本当に無いのか
- 本当は有るのに、
  サーバが嘘をついているのか

クライアントは判断できない

クライアントは、
辞書を持たずに、
この不正を検出できるか？

最初の解決法

- 竹谷、尾形 (IWSEC 2015)
- サーバが

  「検索ワード $\notin$ 辞書」

  の証明をクライアントに送る。
- ただし、サーバの計算量は

  $O(N \log Nm)$

  N=文書の数、m=キーワードの数

尾形、黒澤 (FC 2017)

Passiveな敵に対し安全な
任意のSSE方式

変換方法

辞書不要で
Activeな敵に対しUC安全なSSE方式

「検索ワード ∉ 辞書」
の証明に必要なサーバの計算量 O(1)

たとえば

Asharovらの方式 (CRYPTO 2018)

Passiveな敵に対し安全な
任意のSSE方式

本方法

辞書不要で
Activeな敵に対しUC安全なSSE方式

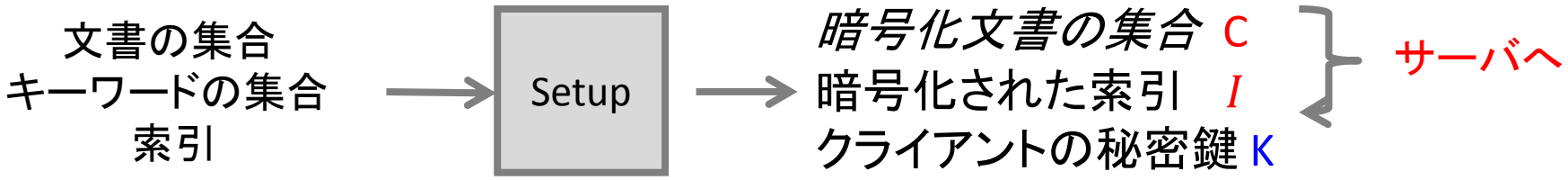
効率のよいUC安全なSSE方式

SSE方式の定義

- Setupアルゴリズム
- Trpdr（トラップドア）アルゴリズム
- Searchアルゴリズム
- Dec（復号）アルゴリズム

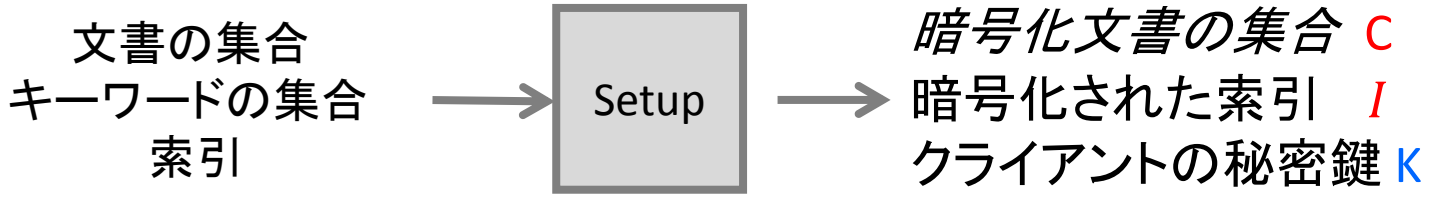
SSE方式

保存フェーズ

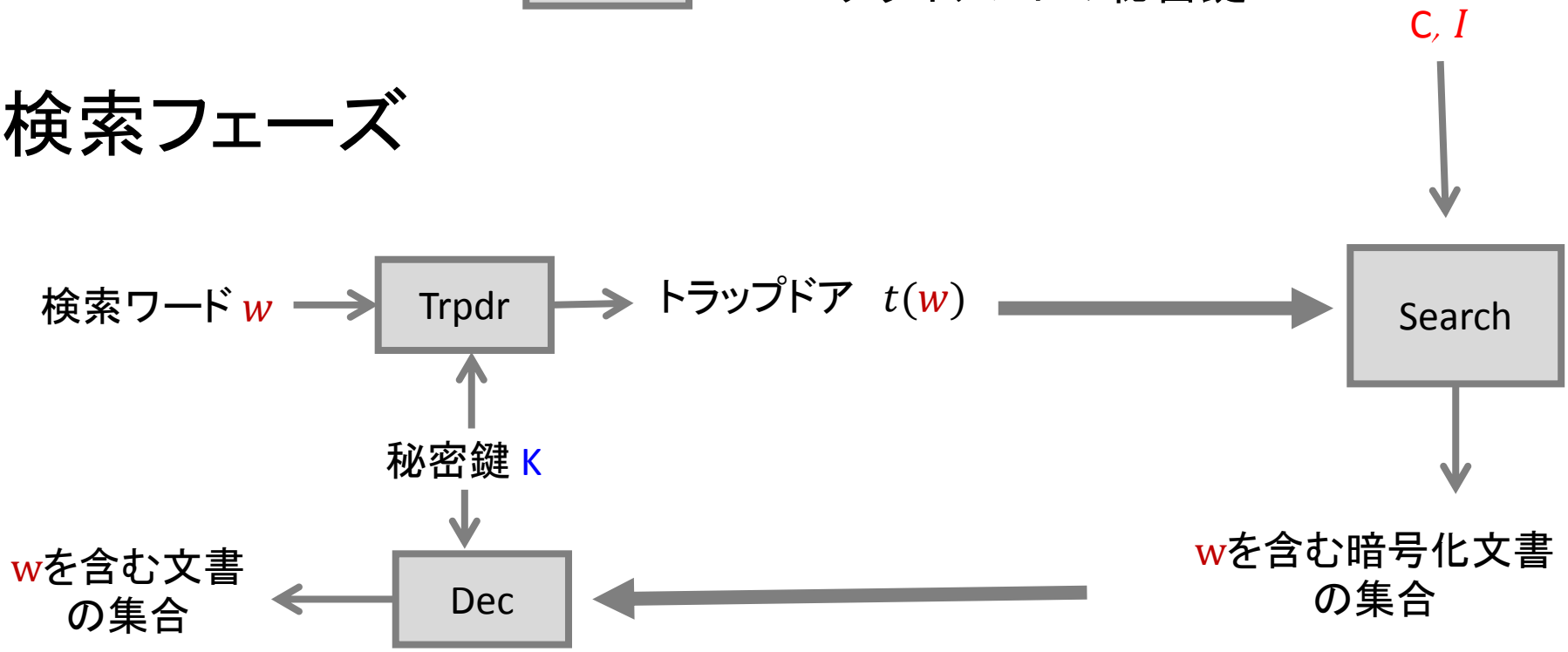


SSE方式

保存フェーズ



検索フェーズ



尾形、黒澤 (FC 2017)

Passiveな敵に対し安全な
任意のSSE方式

変換方法

辞書不要で
Activeな敵に対し安全なSSE方式

検索ワード $\notin$ 辞書
を証明するのに必要なサーバの計算量 O(1)

我々のメイン ツール

- カッコーハッシュ
- Pagh and Rodler (2001)が開発

- $(x_1, \ldots, x_n)$ を格納するデータ構造
- x_i を取り出すのに、O(1)の手間で済む

$(x_1, \cdots, x_7)$を格納する例

h_1, h_2: ランダムなハッシュ関数

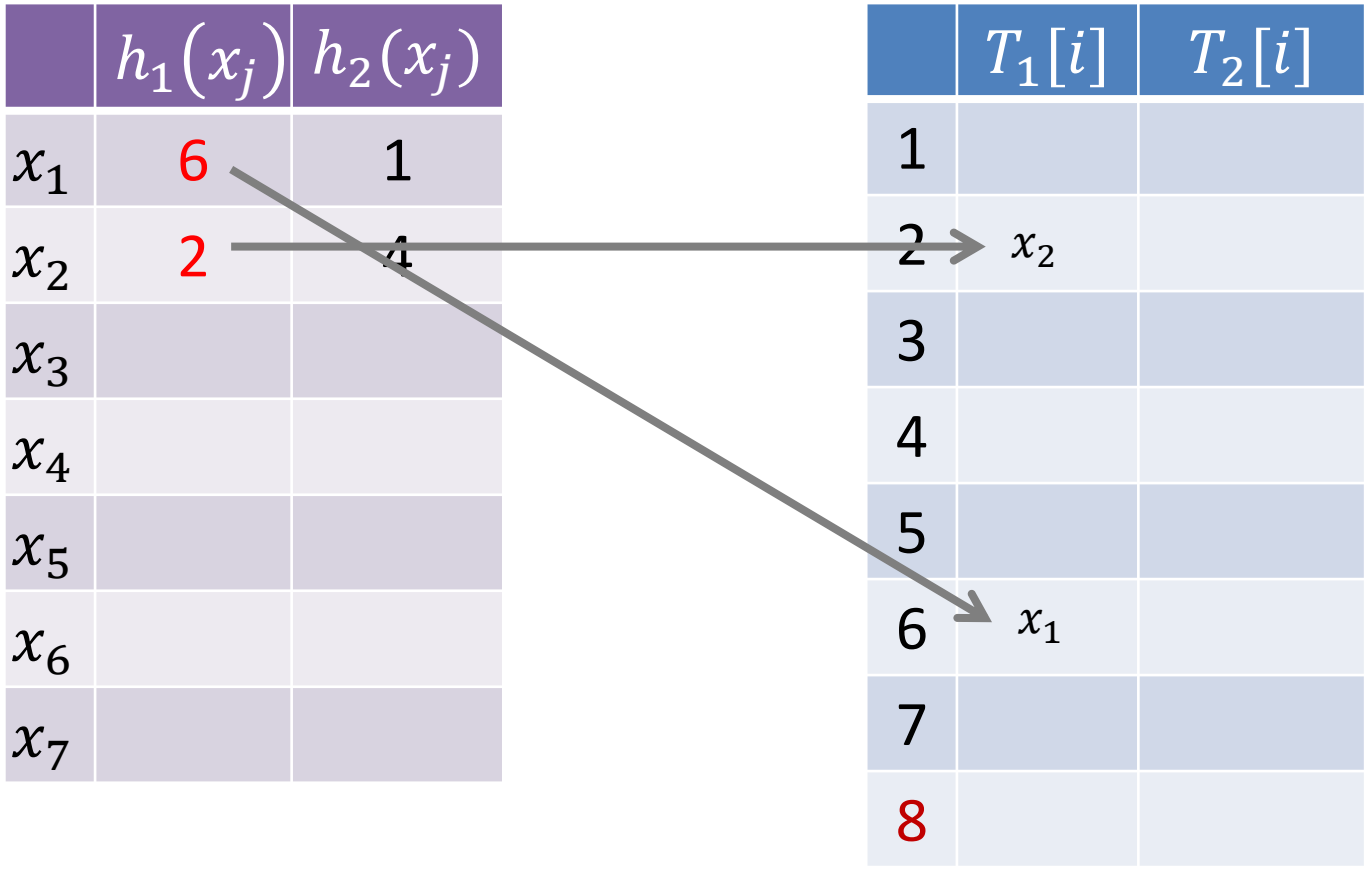
	$h_1(x_j)$	$h_2(x_j)$
x_1	6	1
x_2	2	4
x_3	6	4
x_4	6	3
x_5	7	8
x_6	7	6
x_7	2	8

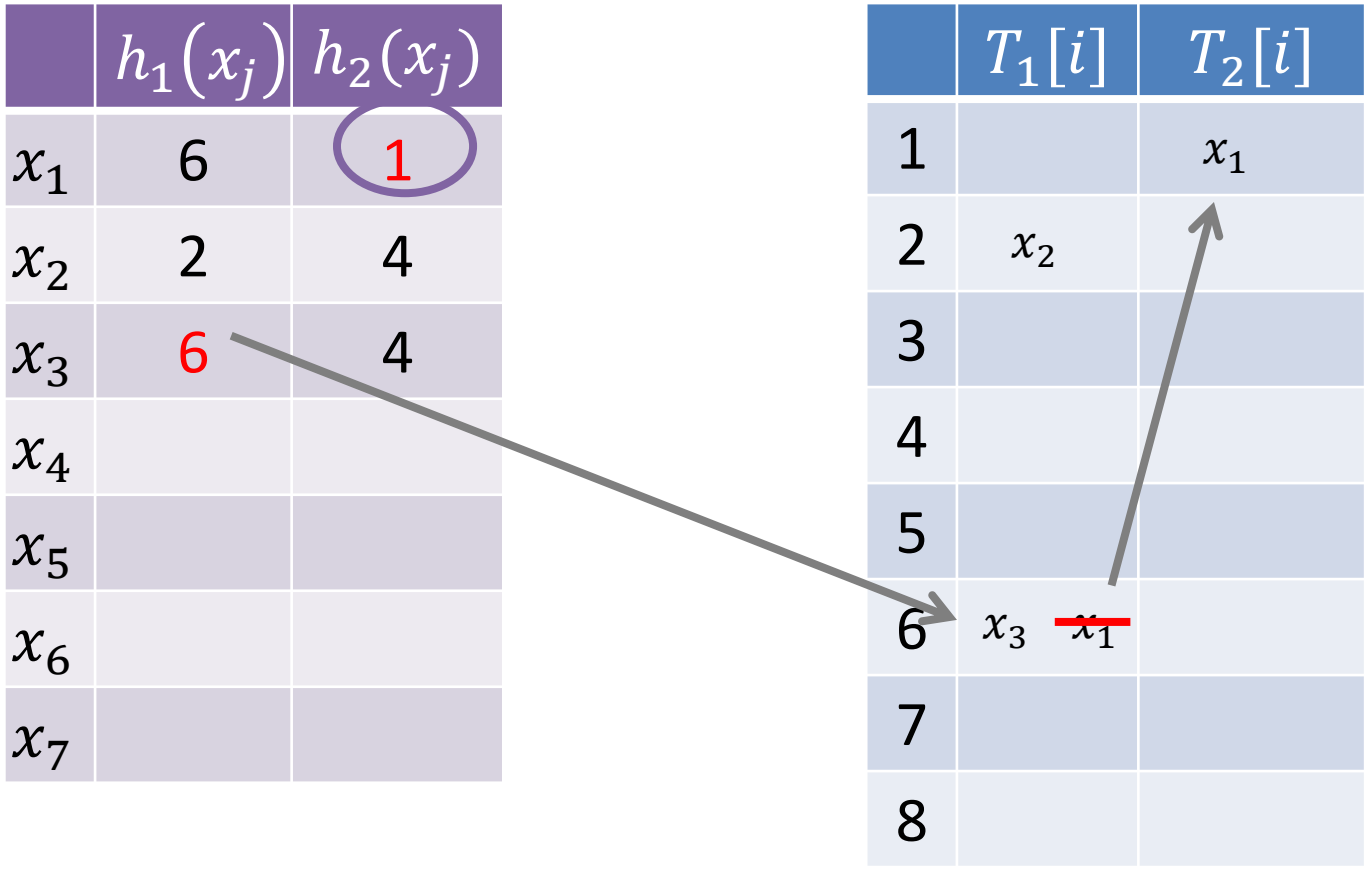
$8 > 7$

	$T_1[i]$	$T_2[i]$
1		
2		
3		
4		
5		
6		
7		
8		

$(x_1, \cdots, x_7)$ を格納する例

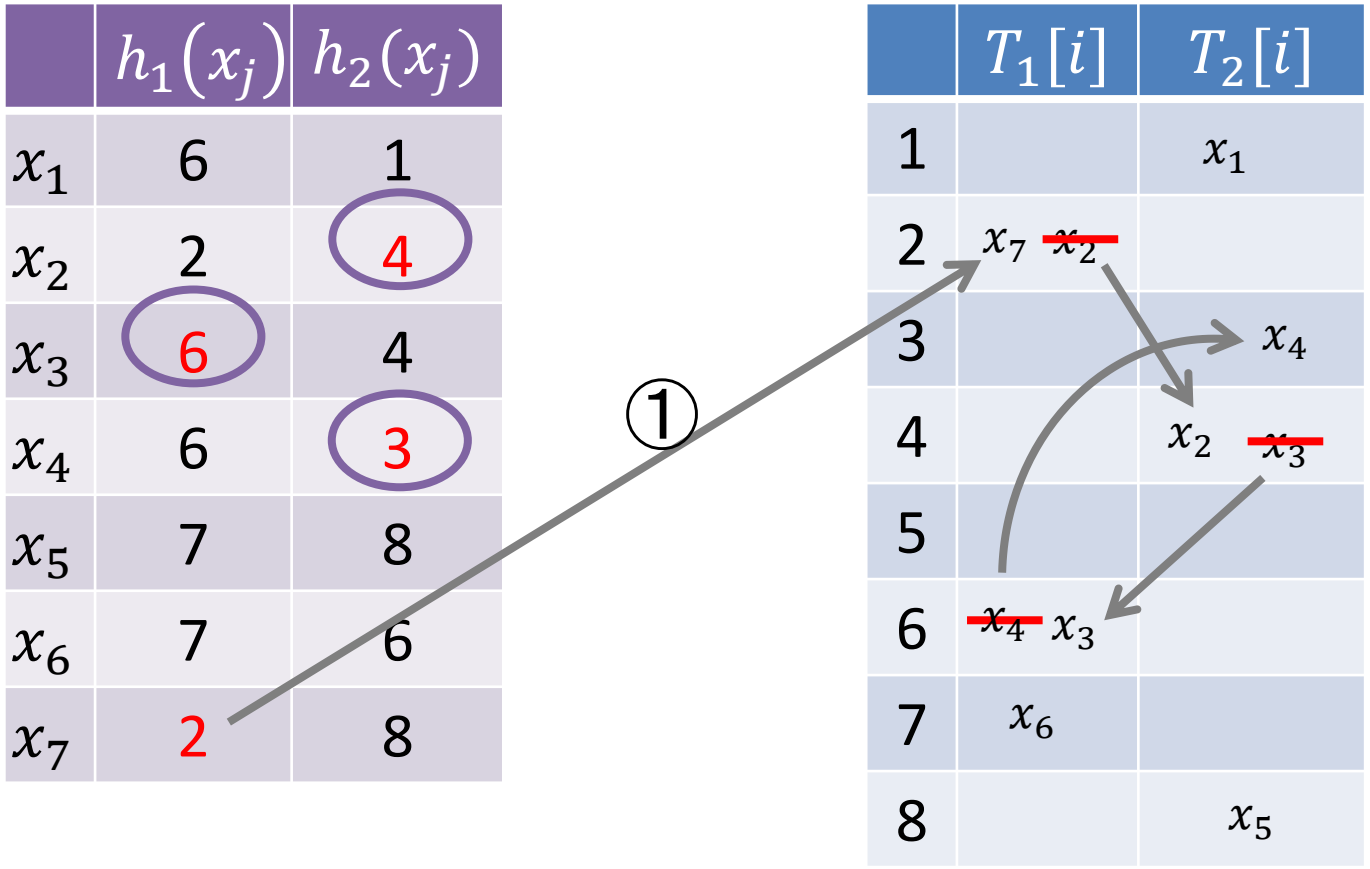
$m = 8$

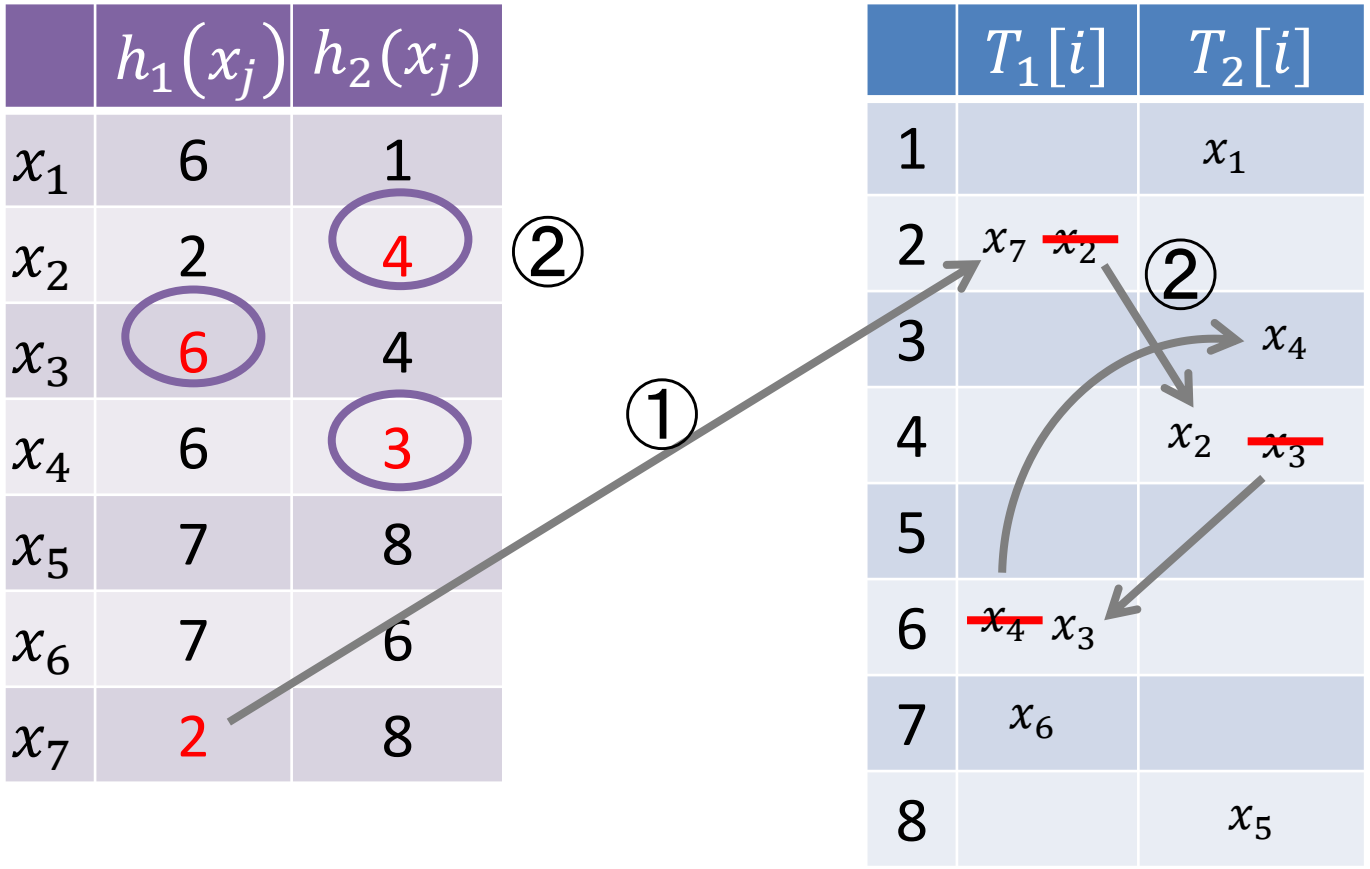


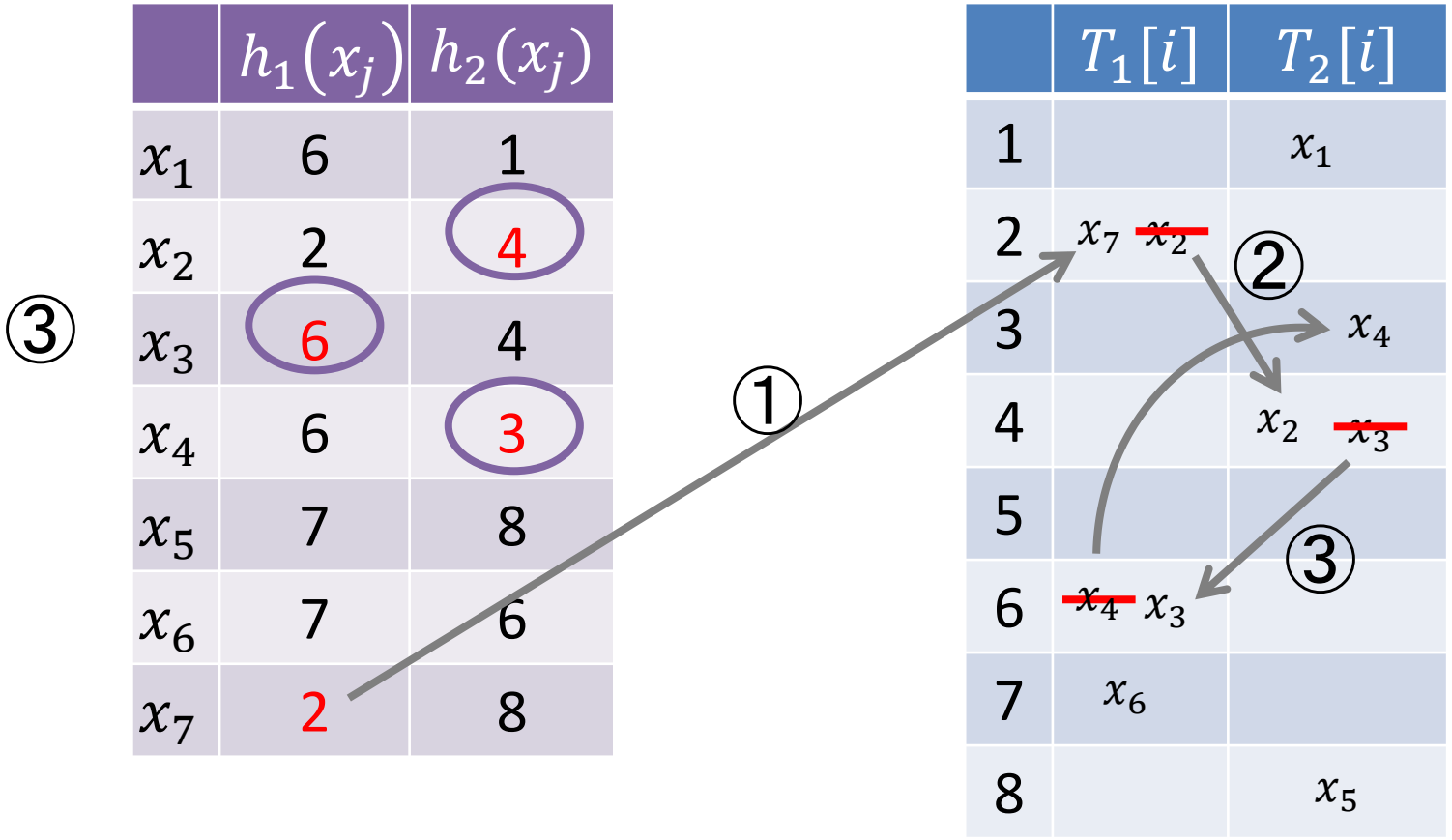

$h_1(x_j)$
$h_2(x_j)$
x_1 6 1
x_2 2 4
x_3 6 4
x_4
x_5
x_6
x_7
$T_1[i]$
$T_2[i]$
1 x_1
2 x_2
3
4
5
6 x_3 ~~x_1~~
7
8

	$h_1(x_j)$	$h_2(x_j)$
x_1	6	1
x_2	2	4
x_3	6	4
x_4	6	3
x_5	7	8
x_6	7	6
x_7		

	$T_1[i]$	$T_2[i]$
1		x_1
2	x_2	
3		
4		x_3
5		
6	x_4	
7	x_6	
8		x_5

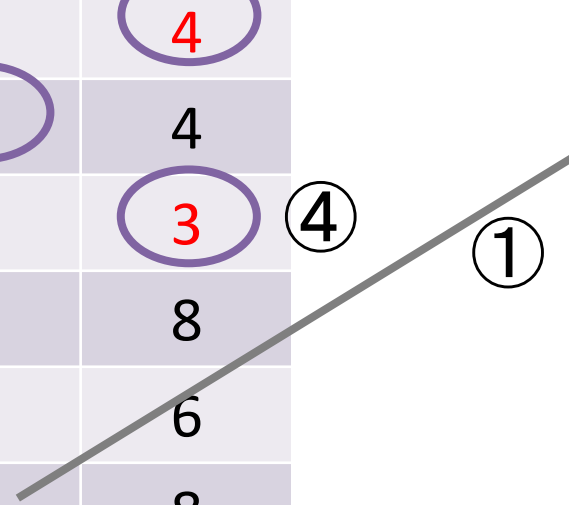

$h_1(x_j)$
$h_2(x_j)$
x_1 6 1
x_2 2 4
x_3 6 4
x_4 6 3
x_5 7 8
x_6 7 6
x_7 2 8
①
$T_1[i]$
$T_2[i]$
1 x_1
2 x_7 ~~x_2~~
3 x_4
4 x_2 ~~x_3~~
5
6 ~~x_4~~ x_3
7 x_6
8 x_5


$h_1(x_j)$
$h_2(x_j)$
x_1 6 1
x_2 2 4
x_3 6 4
x_4 6 3
x_5 7 8
x_6 7 6
x_7 2 8
①
②
$T_1[i]$
$T_2[i]$
1 x_1
2 x_7 ~~x_2~~
3 x_4
4 x_2 ~~x_3~~
5
6 ~~x_4~~ x_3
7 x_6
8 x_5
②


③
$h_1(x_j)$
$h_2(x_j)$
x_1 6 1
x_2 2 4
x_3 6 4
x_4 6 3
x_5 7 8
x_6 7 6
x_7 2 8
①
$T_1[i]$
$T_2[i]$
1 x_1
2 x_7 ~~x_2~~
②
3 x_4
4 x_2 ~~x_3~~
5
③
6 ~~x_4~~ x_3
7 x_6
8 x_5

	$h_1(x_j)$	$h_2(x_j)$
x_1	6	1
x_2	2	4
x_3	6	4
x_4	6	3
x_5	7	8
x_6	7	6
x_7	2	8

	$T_1[i]$	$T_2[i]$
1		x_1
2	x_7 ~~x_2~~	
3		x_4
4		x_2 ~~x_3~~
5		
6	~~x_4~~ x_3	
7	x_6	
8		x_5



終了

適当な回数やってダメだったら

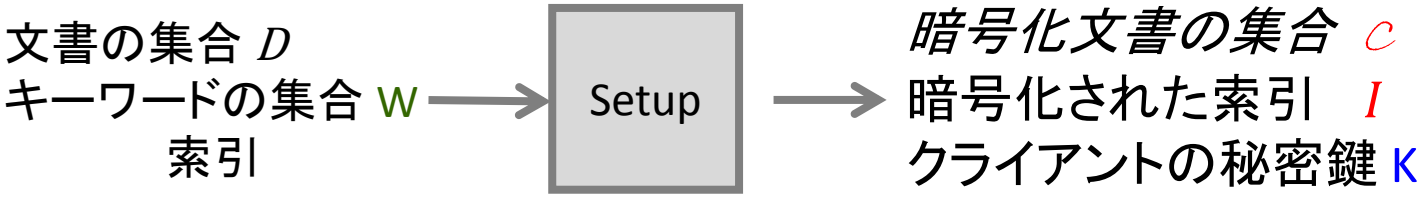
- ハッシュ関数h_1, h_2をランダムに選び直し、再度実行
- それでも、

  計算時間の期待値＝O(n)

  ただし、n=データ数

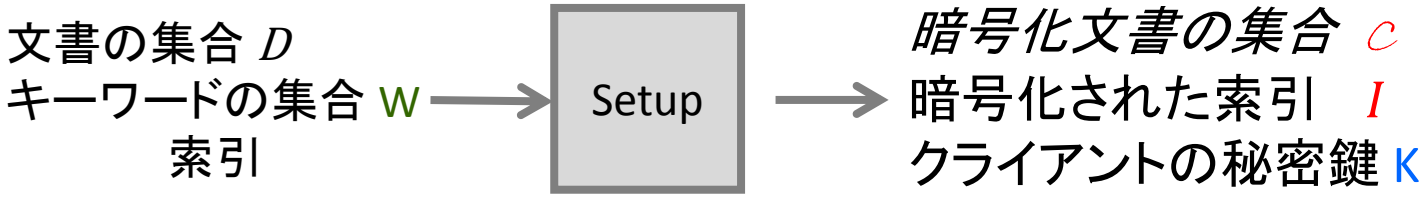
提案する変換方法

元のSSE方式のSetupアルゴリズムを走らせる



提案する変換方法

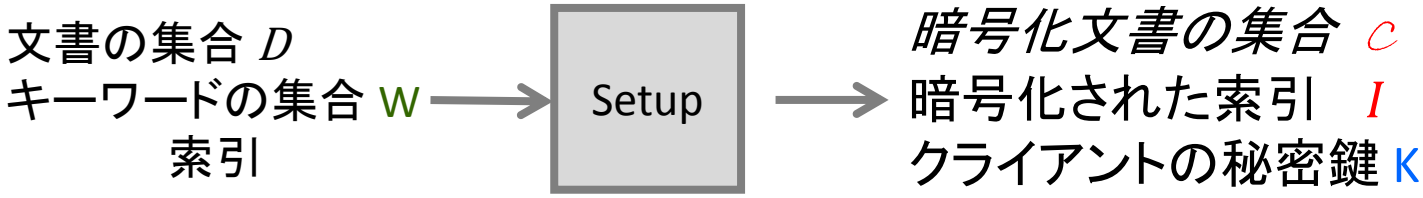
元のSSE方式のSetupアルゴリズムを走らせる



の鍵 k をランダムに選ぶ。

提案する変換方法

元のSSE方式のSetupアルゴリズムを走らせる



の鍵 k をランダムに選ぶ。

岩田、黒澤のCMACは、そのような擬似ランダム関数

各キーワードに対し

X=F_k(福島）
Y=F_k(茨城）
Z=F_k(ハワイ）

ハッシュ関数
h_1, h_2
テーブル
T_1, T_2

T_1=

Z
X

T_2=

Y

2列目

T_1=

	$a_1=F_k(T_1$の1行目は空）
Z	$a_2=F_k(T_1$の2行目はZ）
X	$a_3=F_k(T_1$の3行目はX）

T_2=

	$a_1'=F_k(T_2$の1行目は空）
	$a_2'=F_k(T_2$の2行目は空）
Y	$a_3'=F_k(T_2$の3行目はY）

３列目

C_1=E(文書1)、…、C_5=E(文書5）

T_1=

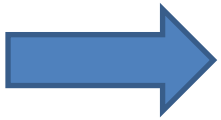
	a_1	
Z	a_2	$b_2=F_k$(Zへの応答はC_1, C_2, C_3)
X	a_3	$b_3=F_k$(Xへの応答はC_1, C_3, C_5)

T_2=

	a_1'	
	a_2'	
Y	a_3'	$b_3'=F_k$(Yへの応答はC_2, C_4)

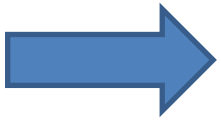
クライアント

暗号化文書の集合 C
暗号化された索引 I
カッコー ハッシュ・テーブルT_1, T_2
ハッシュ関数 h_1, h_2



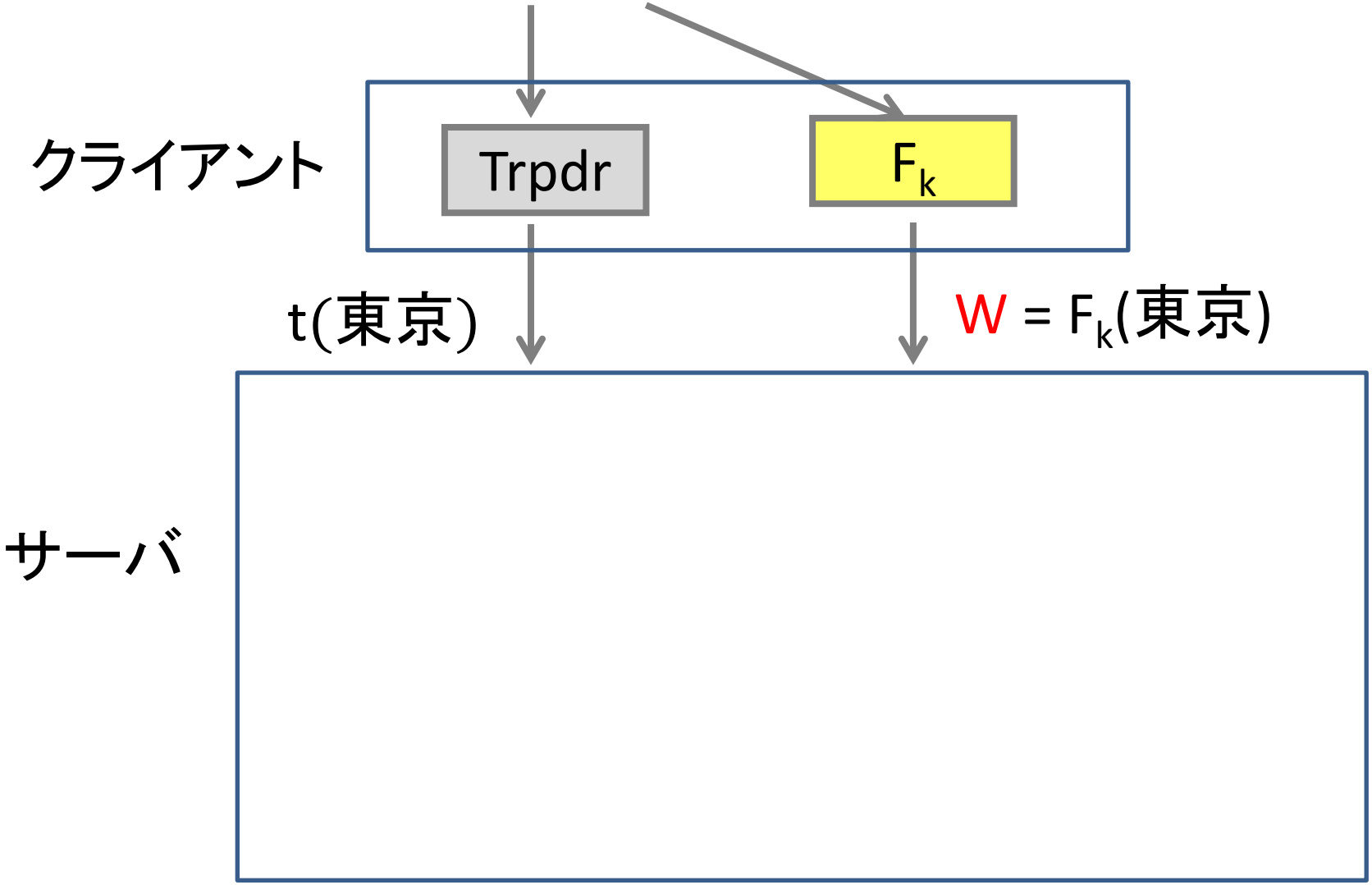
サーバ

元のSSE方式の秘密鍵 K
擬似ランダム関数の鍵 k
ハッシュ関数 h_1, h_2

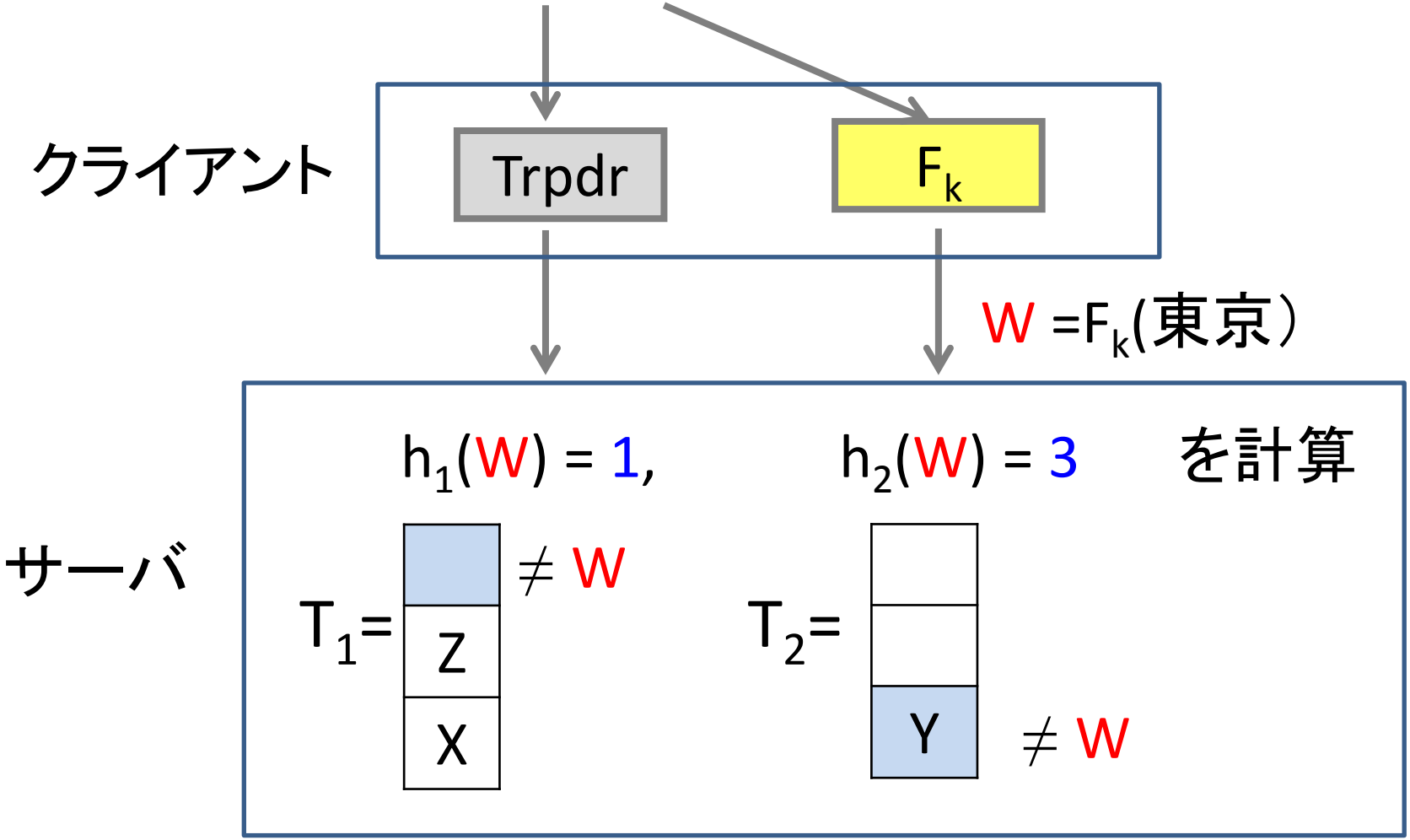


自分のPC

(1) 東京で検索するとき



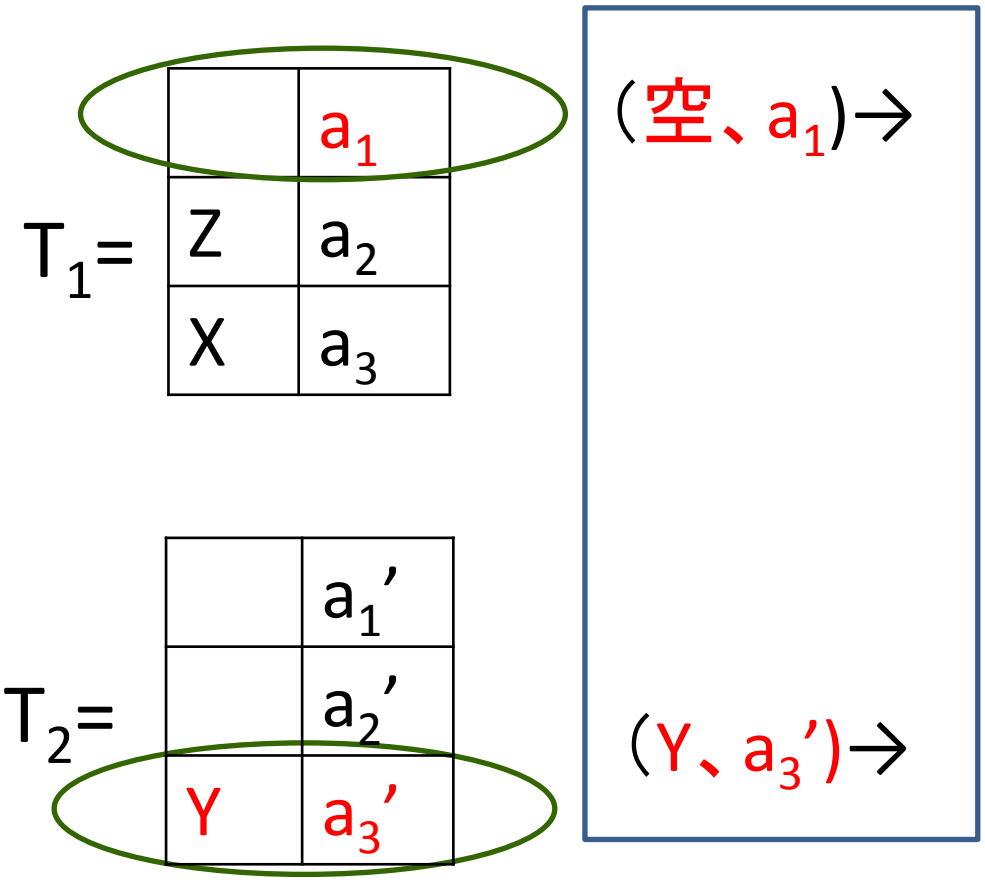
(1) 東京で検索するとき



O(1)で、検索ワード ∉ 辞書とわかる

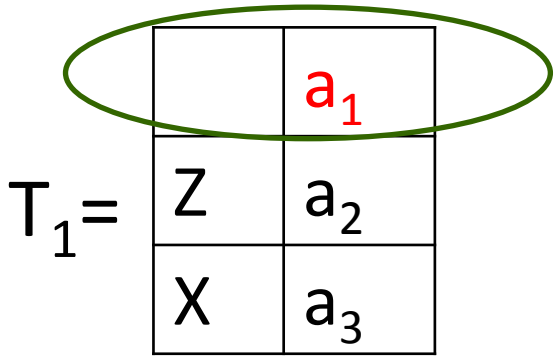
次に

サーバ　　クライアント



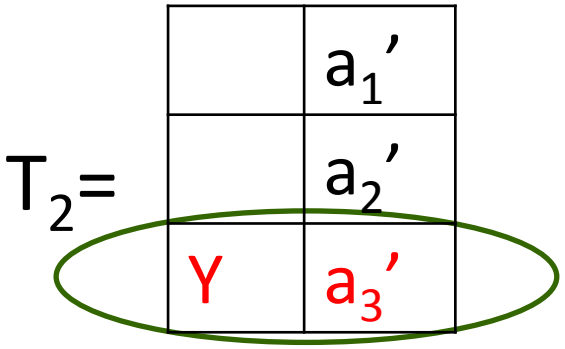
この手間も、O(1)

サーバ　　クライアント



（空、a_1）→　$a_1=F_k(T_1$の1行目は空）をチェック

本当に、
「検索ワード∉辞書」
を納得



Y≠W かつ

（Y、a_3'）→　$a_3'=F_k(T_2$の3行目はY）をチェック

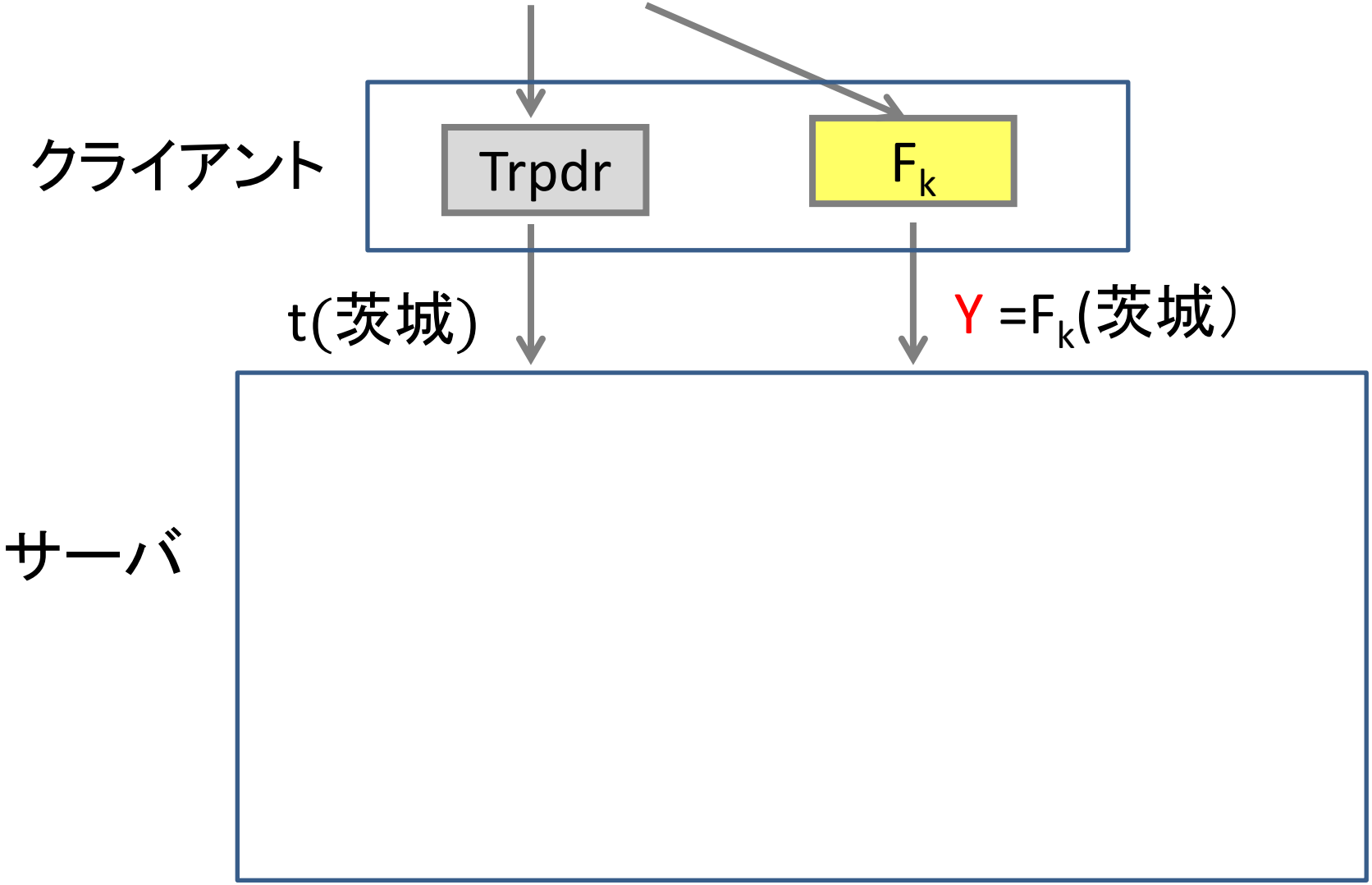
以上、

- サーバは O(1)で

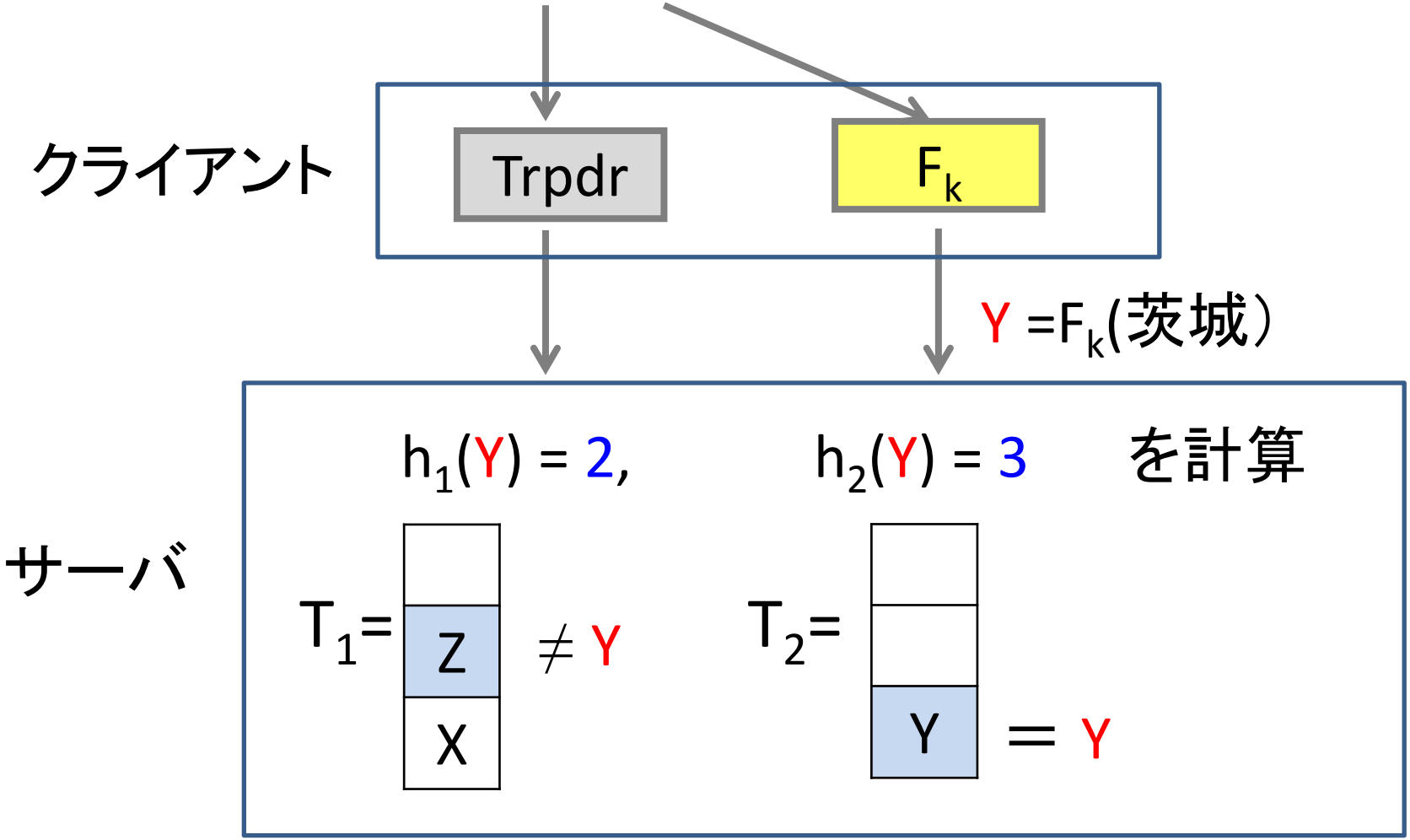
  検索ワード ∉ キーワード集合

- を証明できた。

(2) 茨城で検索するとき



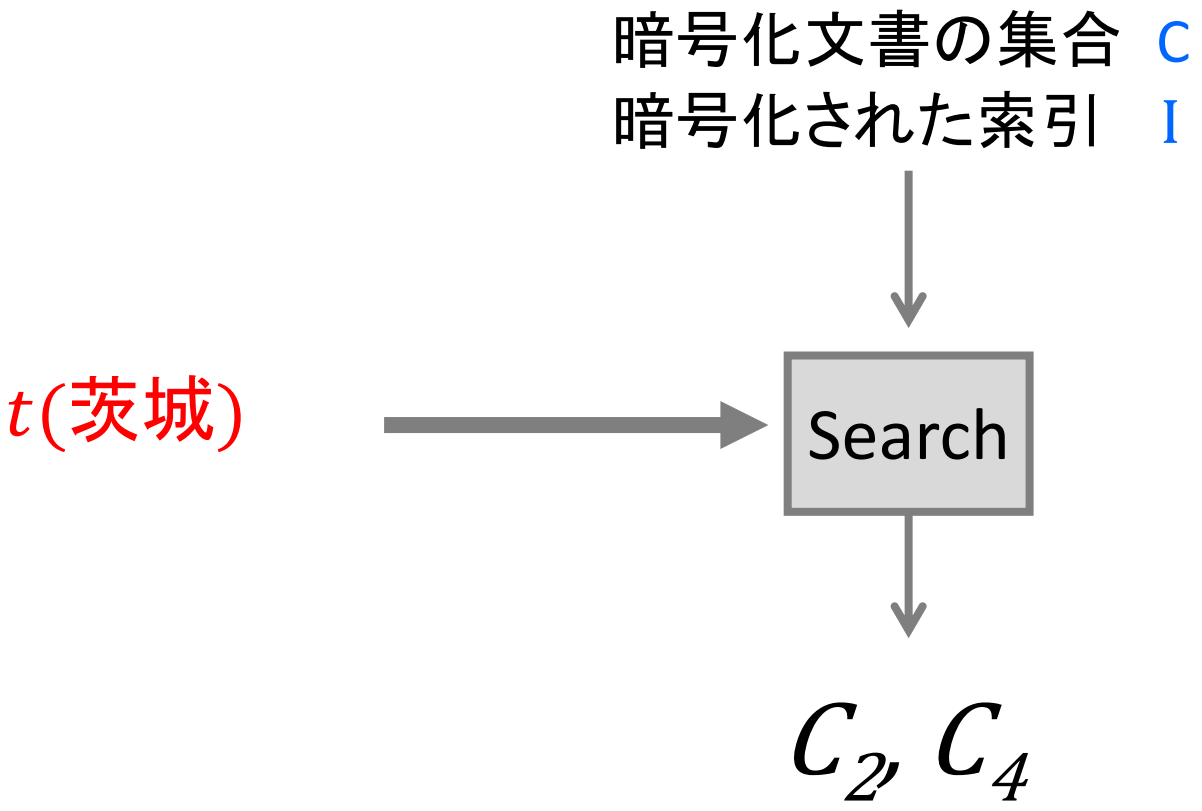
(2) 茨城で検索するとき



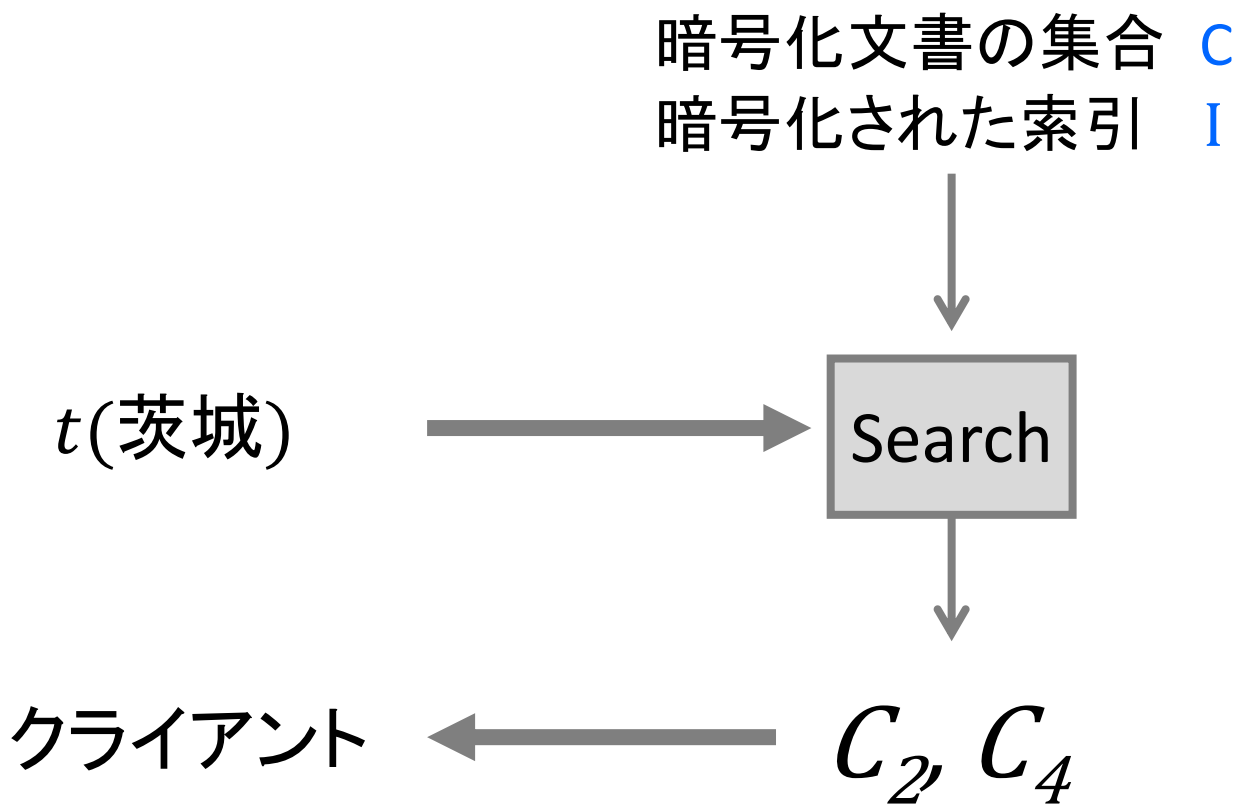
O(1)で、検索ワード ∈ 辞書とわかる

このとき、サーバは、

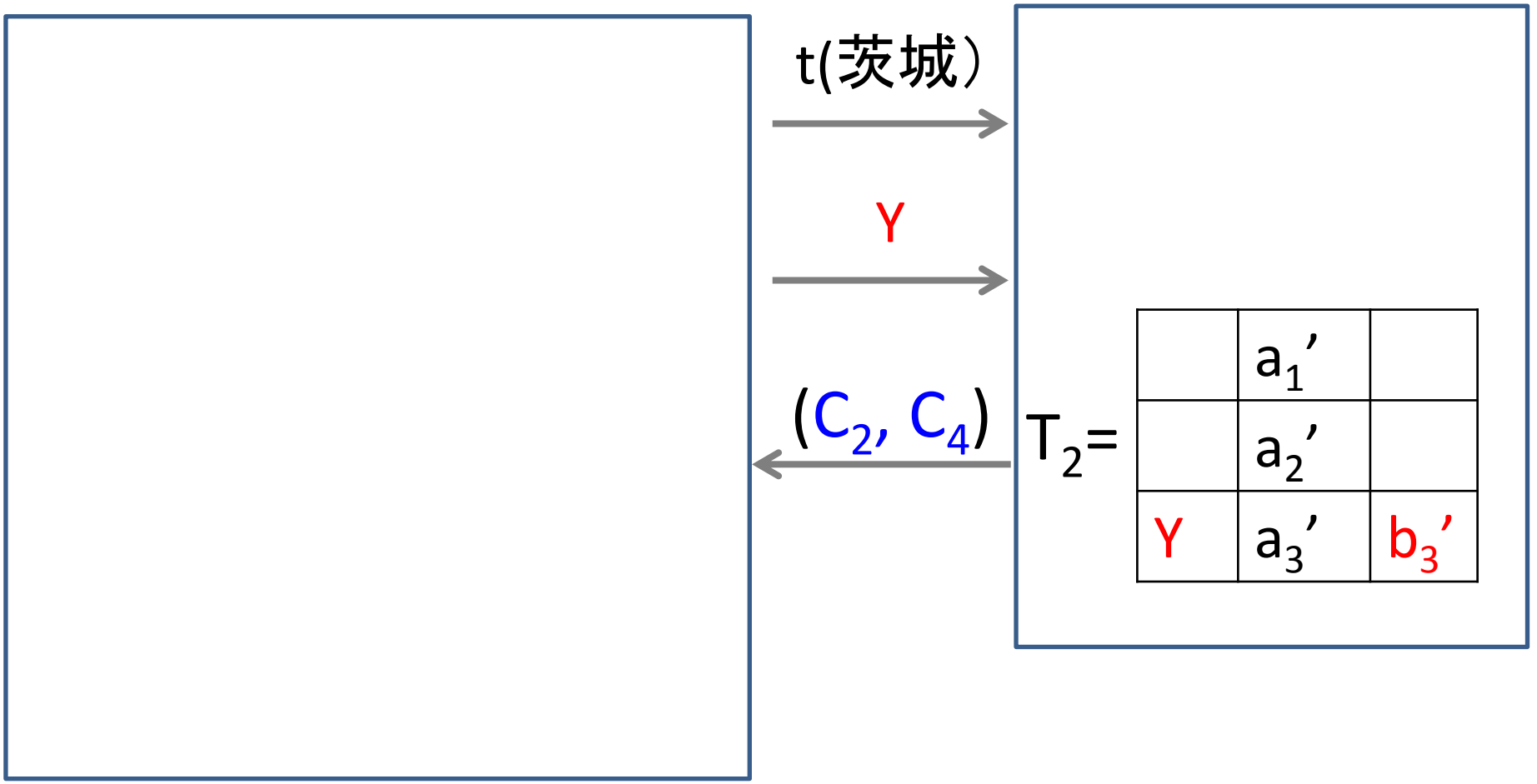
元のSSE方式のSearchアルゴリズムを走らせる



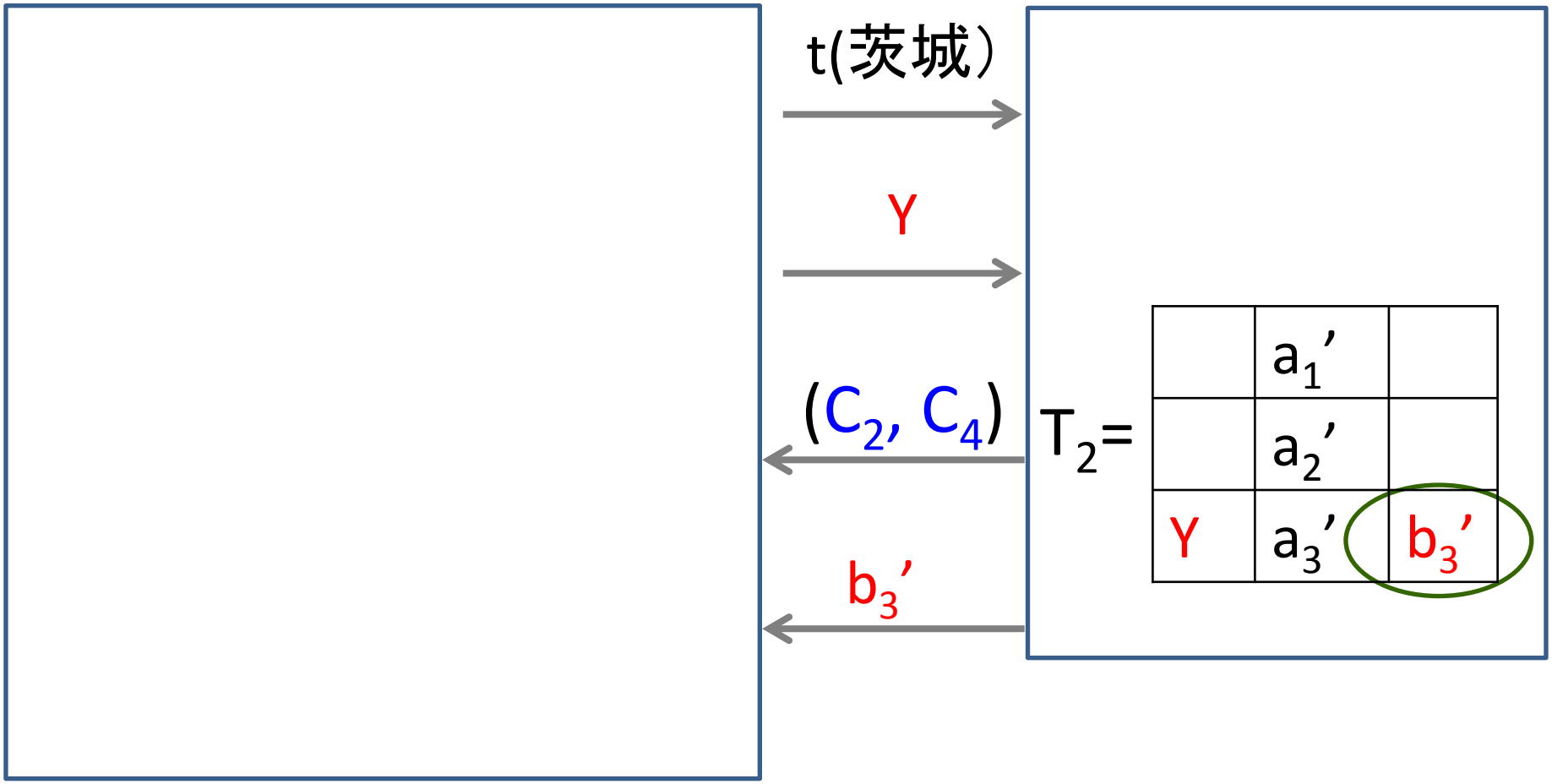
サーバは、



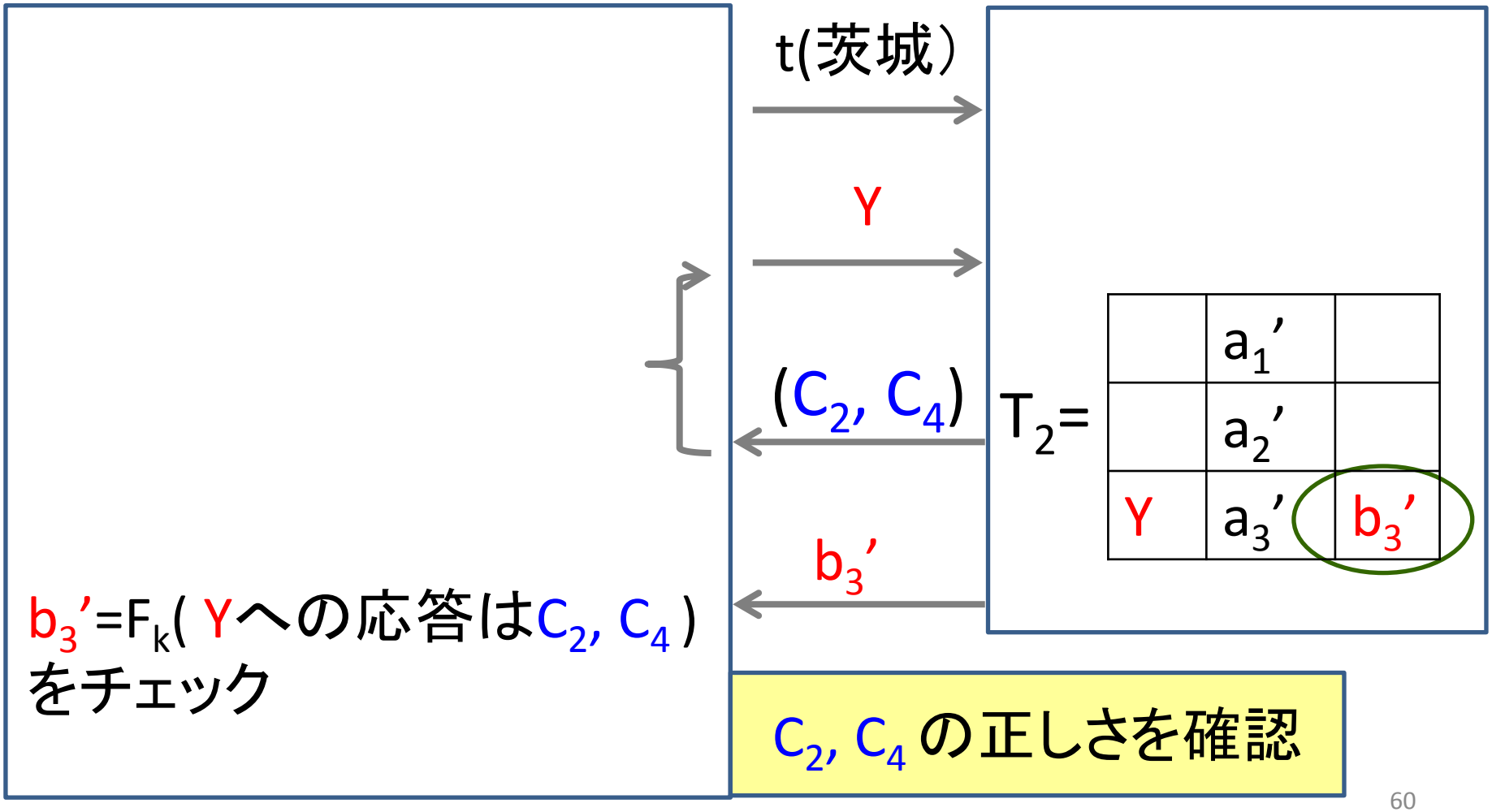
クライアント　　サーバ



クライアント　　サーバ



クライアント　　サーバ


t(茨城）
Y
(C2, C4)
b3’
T2=
a1’
a2’
Y
a3’
b3’
b3’=Fk(Yへの応答はC2, C4)
をチェック
C2, C4 の正しさを確認

尾形・黒澤 (FC 2017)

Passiveな敵に対し安全な
任意のSSE方式

変換方法

辞書不要で
Activeな敵に対しUC安全なSSE方式

検索ワード $\notin$ 辞書
を証明するのに必要なサーバの計算量 O(1)

省略

- UC安全性の証明
- 辞書なしで検索パターンも秘匿する方式

その他

- AND検索やOR検索が可能なSSE方式

  黒澤 (FC 2014)
- 文書の追加などが可能な dynamic SSE方式

  黒澤、大瀧 (CANS 2013)

  黒澤、太田、米山 (IWSEC 2016)

最後に

- SSEのUC安全性の解説
- Fundamentals Review (2015)
- クラウドストレージサービスにおける安全なキーワード検索（黒澤馨）

Thank you !

Next: 検索パターン

- 東日本大震災以降、Xの検索が増えた。
- すると、サーバに、

  X =「福島」

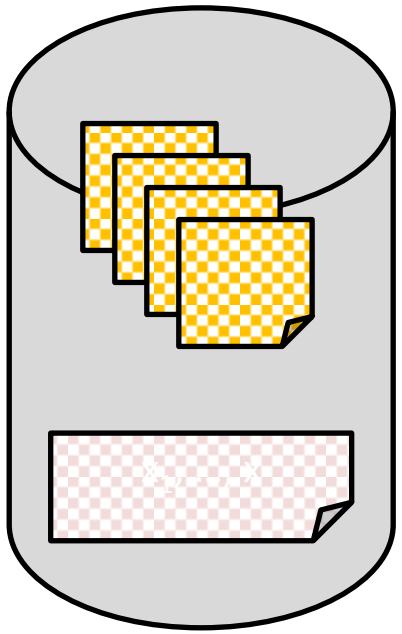
  とわかってしまう。

このような検索パターンも秘匿したい

自明な解決法

サーバ

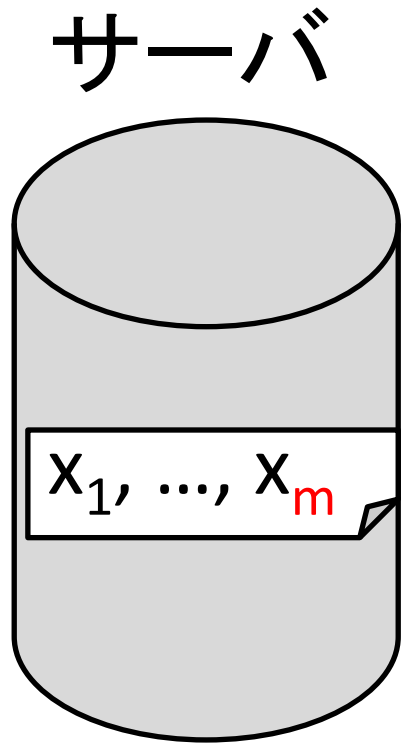
検索要求



格納してあるもの全て

通信量が膨大

Private Information Retrieval (PIR)



i を秘密にしたまま

x_i を取り出す

ただし、
通信量はO(データ数 m)未満

通信量が O((log データ数 m)^2)の方式が存在

SSEへの応用

- クライアントが辞書を覚えていれば
  任意のPIR方式を使って、
  検索パターンを秘匿できる。

- 辞書不要で、これを実現できるか？

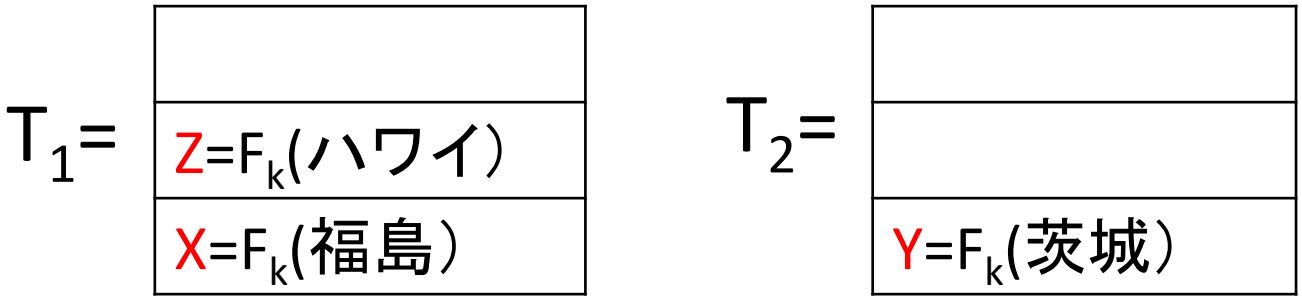
(2) 尾形・黒澤 (accepted by IEICE)

任意のPIR方式

提案方法

辞書不要で
検索パターンを秘匿できる
SSE方式

前回同様、
カッコーハッシュ Table を作成



索引

	文書1	文書2	文書3	文書4	文書5
福島	1	0	1	0	1
茨城	0	1	0	1	0
ハワイ	1	1	1	0	0

T_1=

$Z=F_k$(ハワイ)	E(11100)
$X=F_k$(福島)	E(10101)

T_2=

$Y=F_k$(茨城)	E(01010)

サーバへ格納

茨城で検索するとき

クライアント　　サーバ

$Y=F_k$(茨城)

$h_1(Y) = 2$

T_1の2行目を取り出す

$Z \neq Y$ なので却下

2を秘密にしたまま

PIR

T_1=

Z	E(11100)
X	E(10101)

茨城で検索するとき

クライアント　　サーバ

$Y=F_k$(茨城)

$h_1(Y) = 2$
T_1の2行目を取り出す
$Z \neq Y$ なので却下

$h_2(Y) = 3$
T_2の3行目を取り出す
E(01010)) を復号

2を秘密にしたまま

PIR

3を秘密にしたまま

PIR

T_1=

Z	E(11100)
X	E(10101)

T_2=

Y	E(01010)

茨城で検索するとき

クライアント　　サーバ

$Y=F_k$(茨城)

$h_1(Y) = 2$
T_1の2行目を取り出す
$Z \neq Y$ なので却下

$h_2(Y) = 3$
T_2の3行目を取り出す
E(01010)) を復号

2を秘密にしたまま

PIR

3を秘密にしたまま

PIR

(2, 4)

(C_2, C_4)

T_1=

Z	E(11100)
X	E(10101)

T_2=

Y	E(01010)

(2) 尾形・黒澤 (accepted by IEICE)

任意のPIR方式

提案方法

辞書不要で
検索パターンを秘匿できる
SSE方式