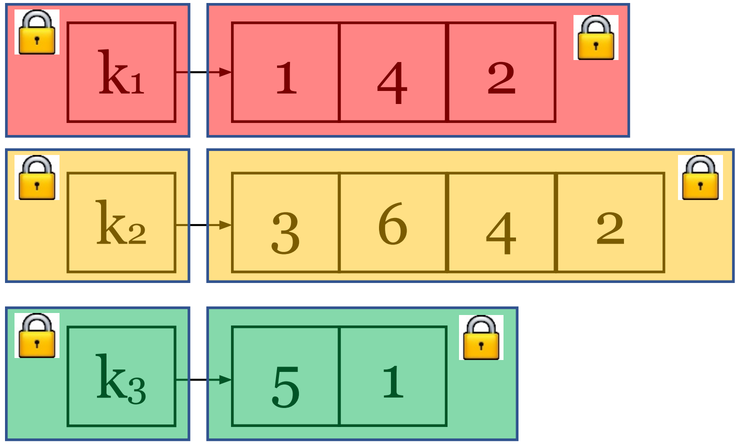
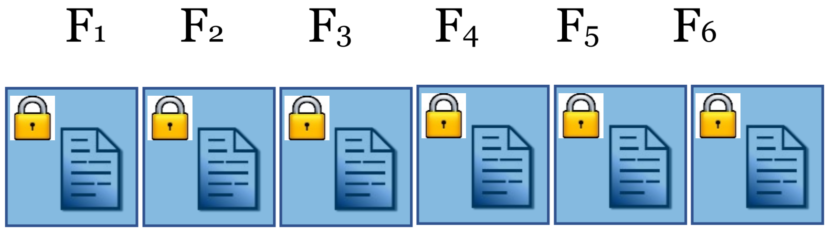
Searchable Symmetric Encryption (SSE)

Encrypted index

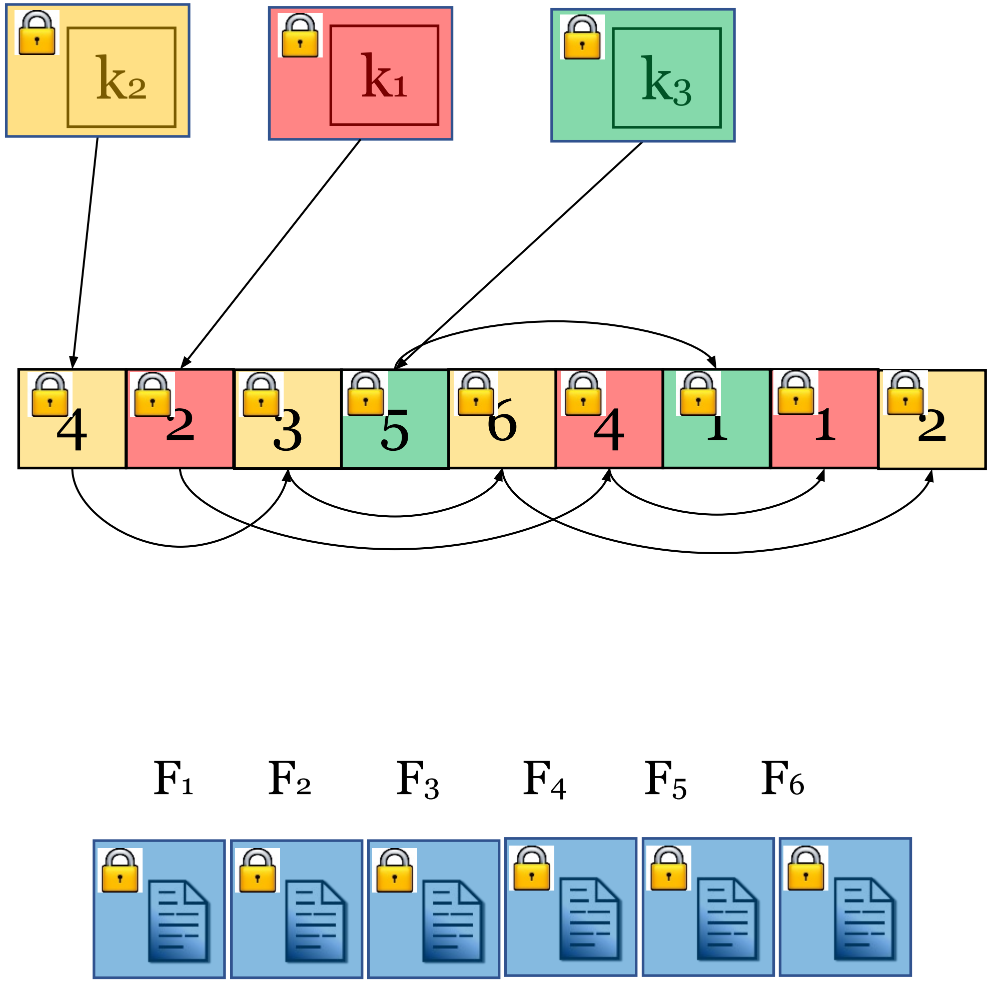
Pseudo random function:



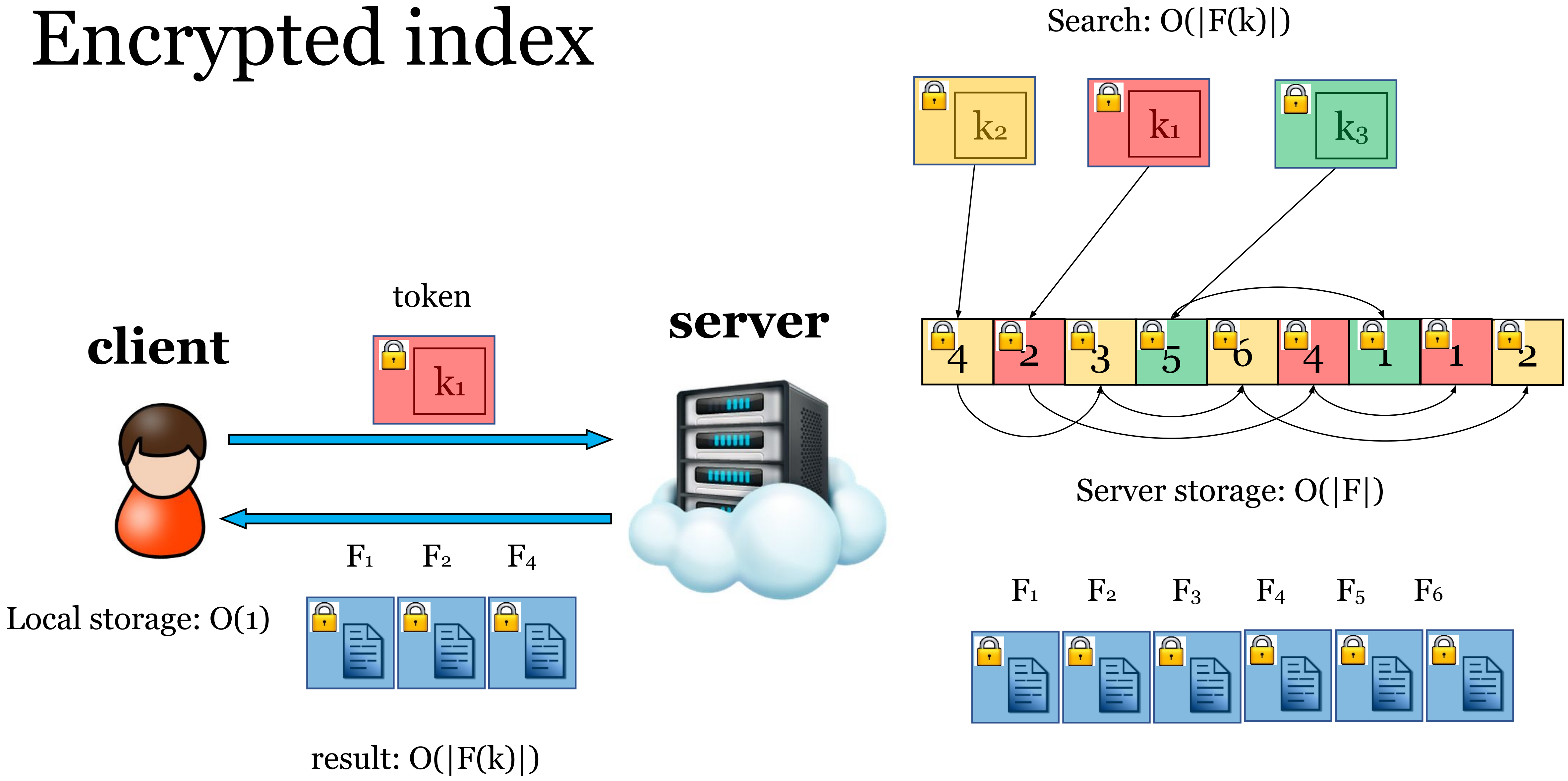
Encryption:



Encrypted index

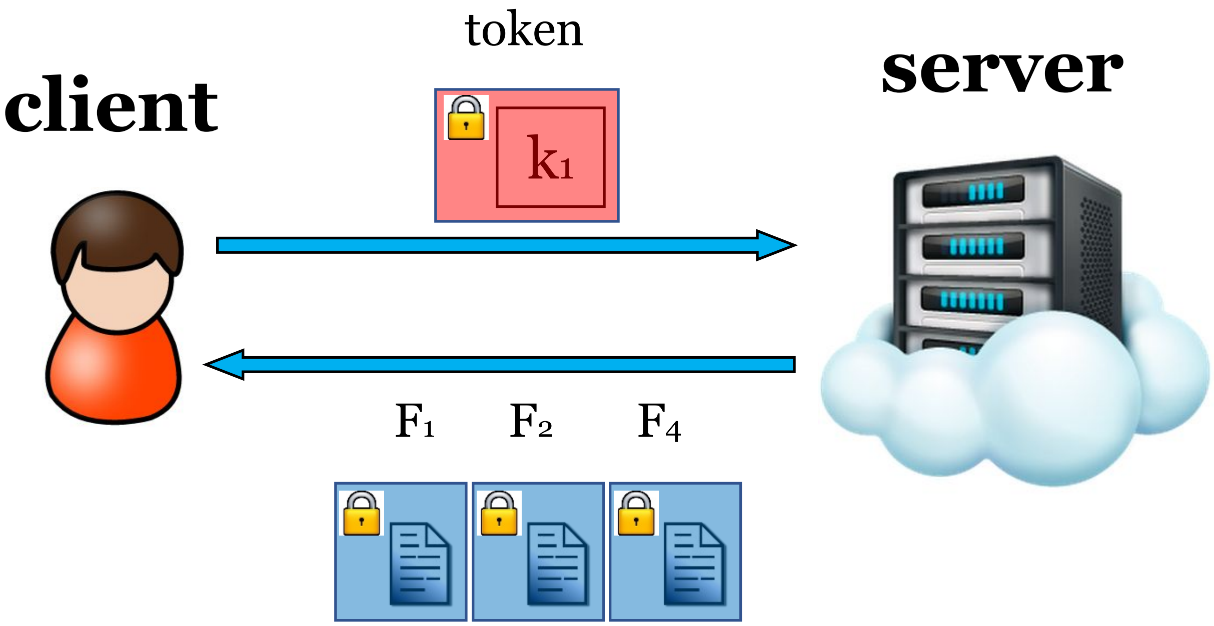


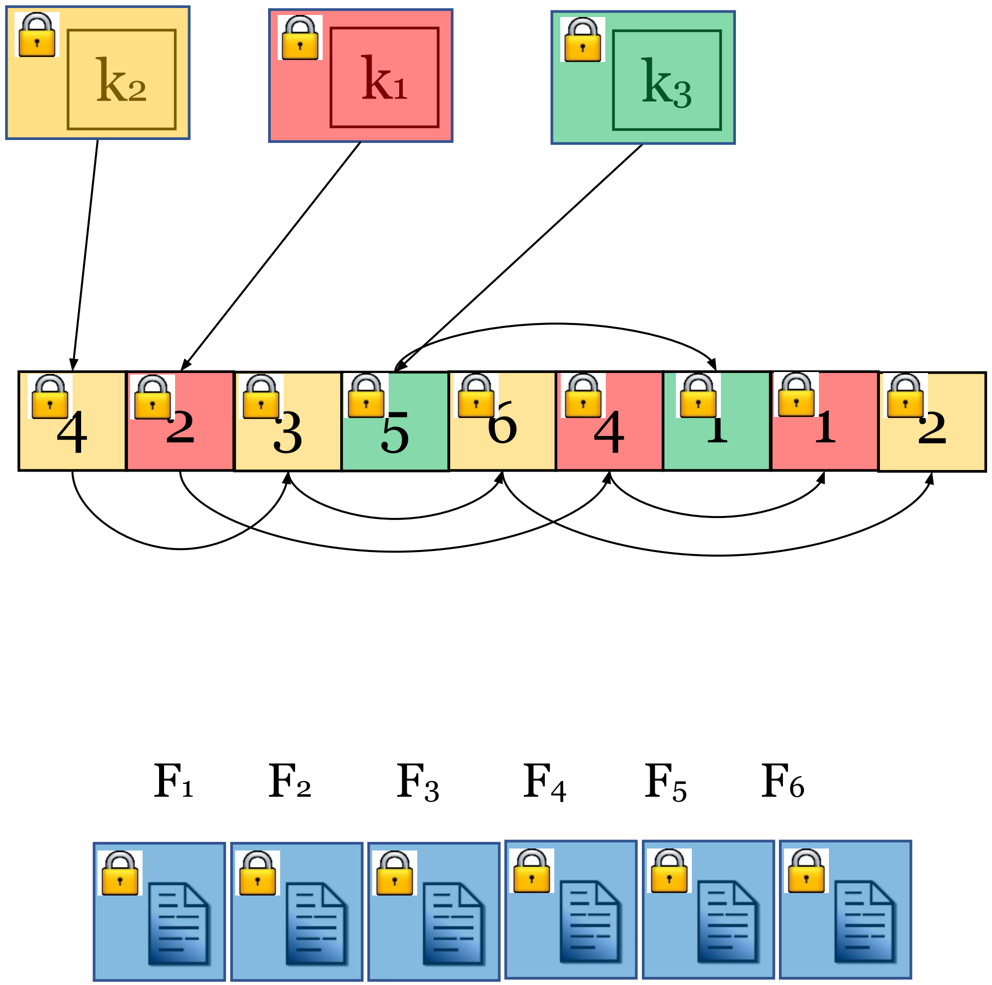
Encrypted index



Encrypted index

deterministic!





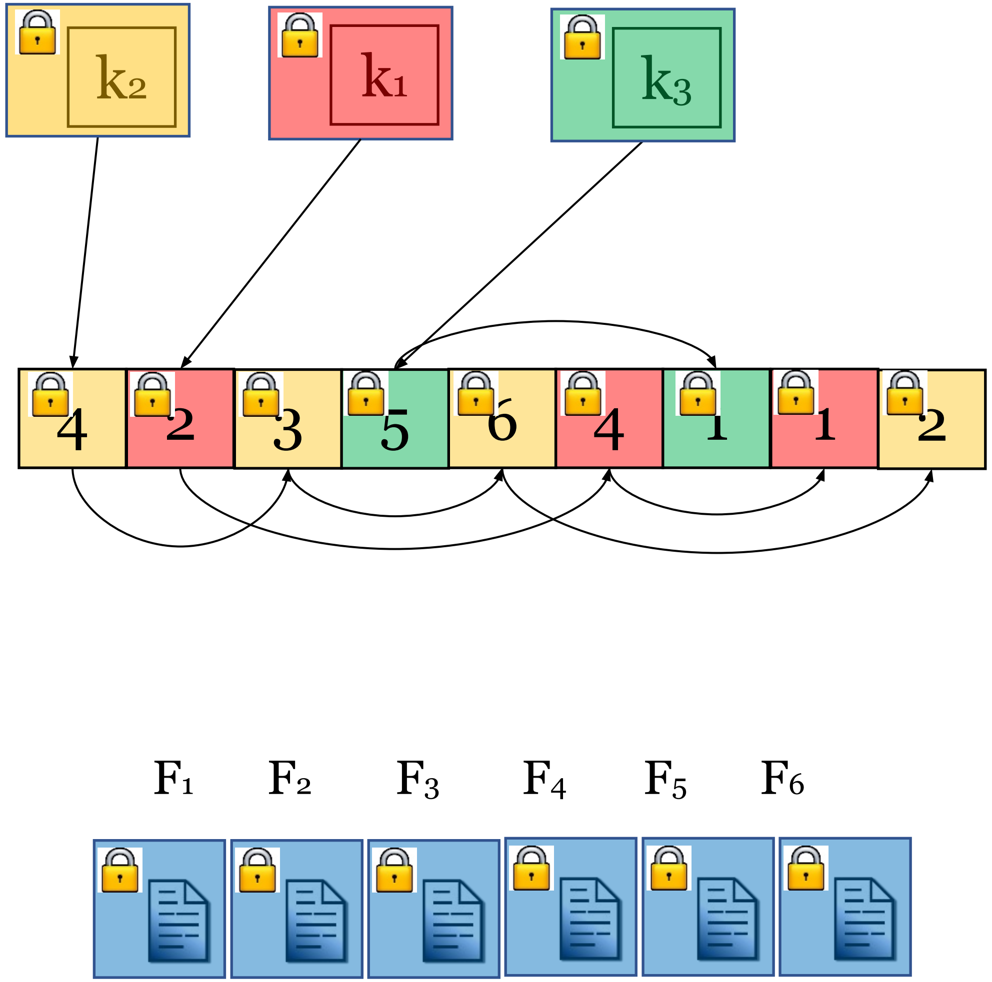
file access patterns!

Dynamic SSE

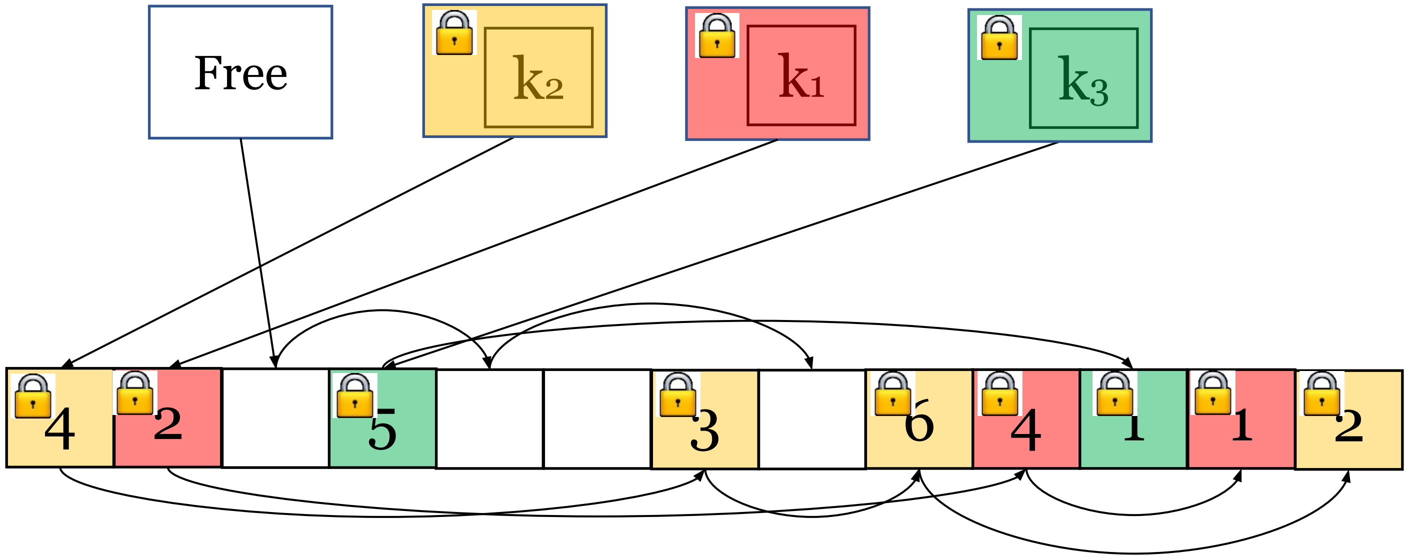
File insertion and deletion

Challenges:

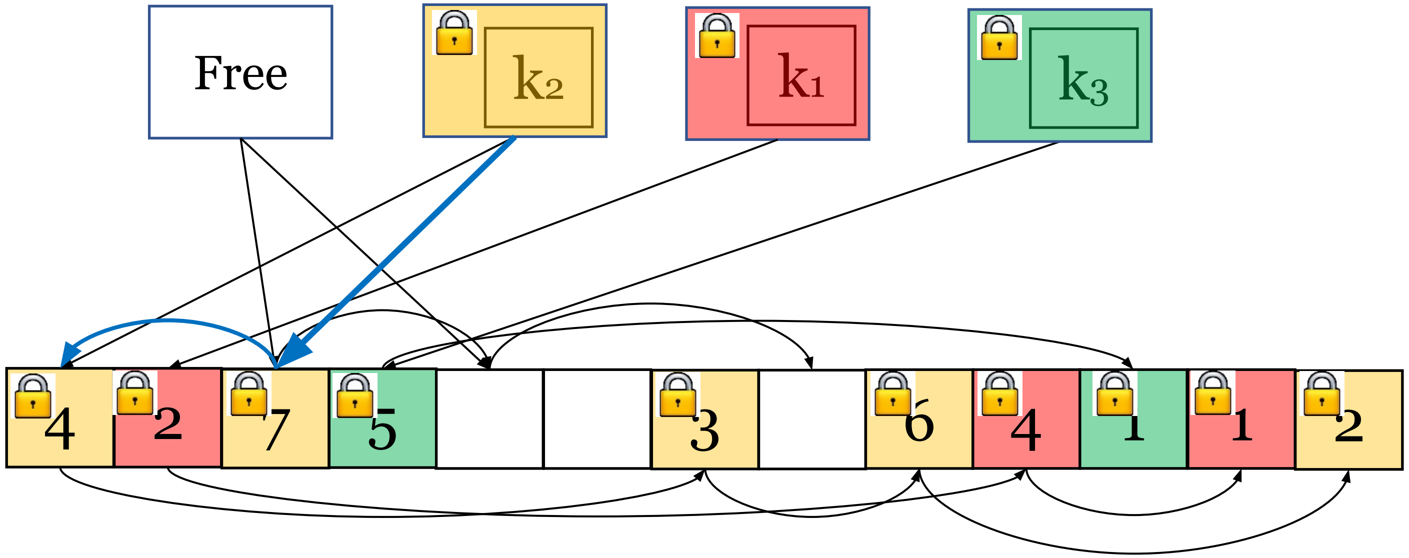
1. Free slot
2. Update previous pointer
3. Removal of (keyword,fileID) pairs



List of free slots



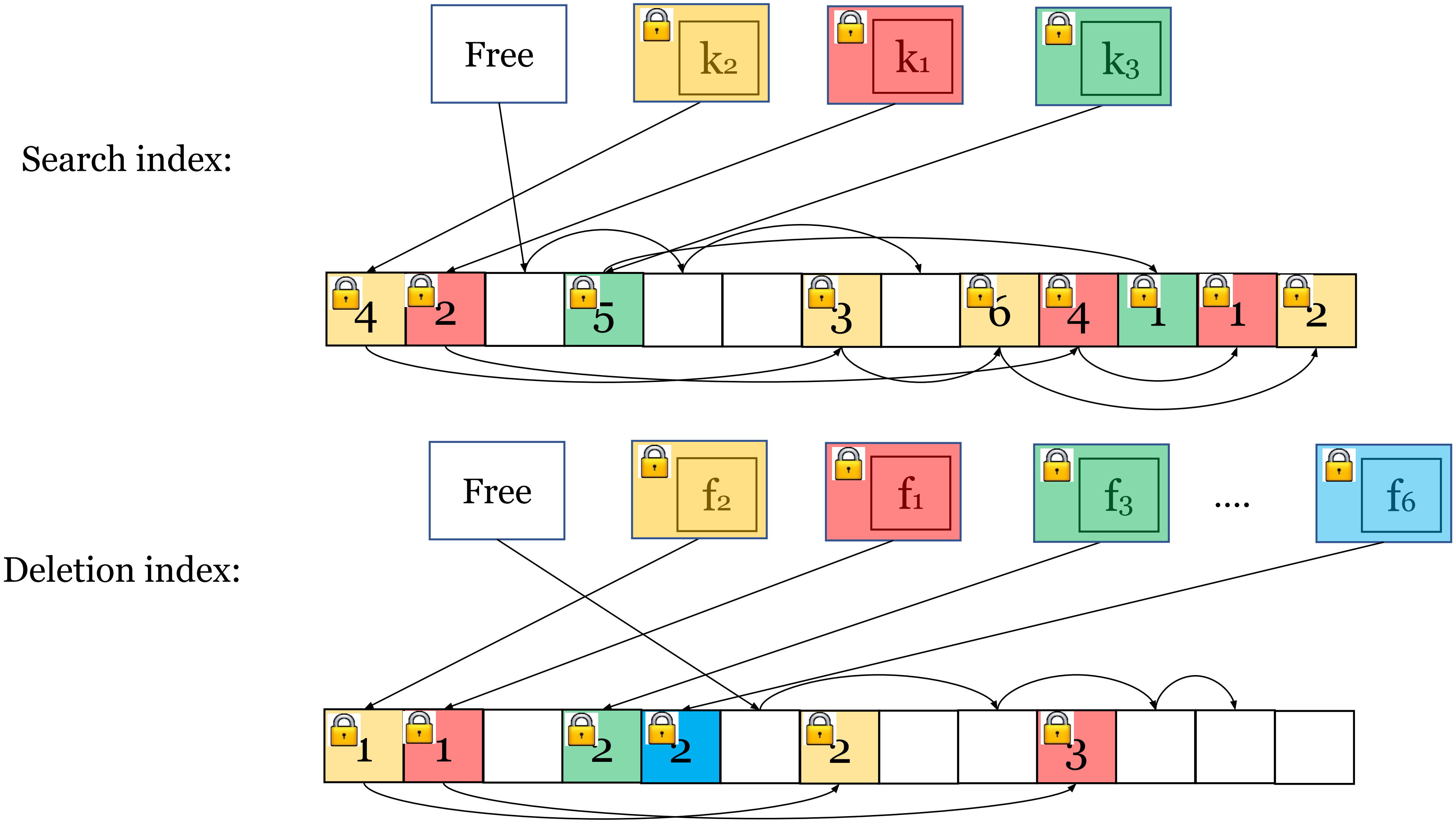
Linearly Homomorphic Pointer



F_7

$\{k_2\}$

Dual Index for Deletion



Dynamic SSE algorithms

- K $\leftarrow$ Gen(1^k)
- $(\gamma, \mathbf{c}) \leftarrow$ Enc(K, $\mathbf{f}$)
- $\tau_s \leftarrow$ SrchToken (K, w)
- $I_w \leftarrow$ Search($\mathbf{c}, \gamma, \tau_s$)
- $f_i \leftarrow$ Dec(K, c_i)

- τ_a,c $\leftarrow$ AddToken (K, f)
- γ',**c**' $\leftarrow$ Add($\mathbf{c}, \gamma, \tau_a$,c)
- $\tau_d \leftarrow$ DelToken (K, f)
- $\gamma' \leftarrow$ Del($\mathbf{c}, \gamma, \tau_d$)

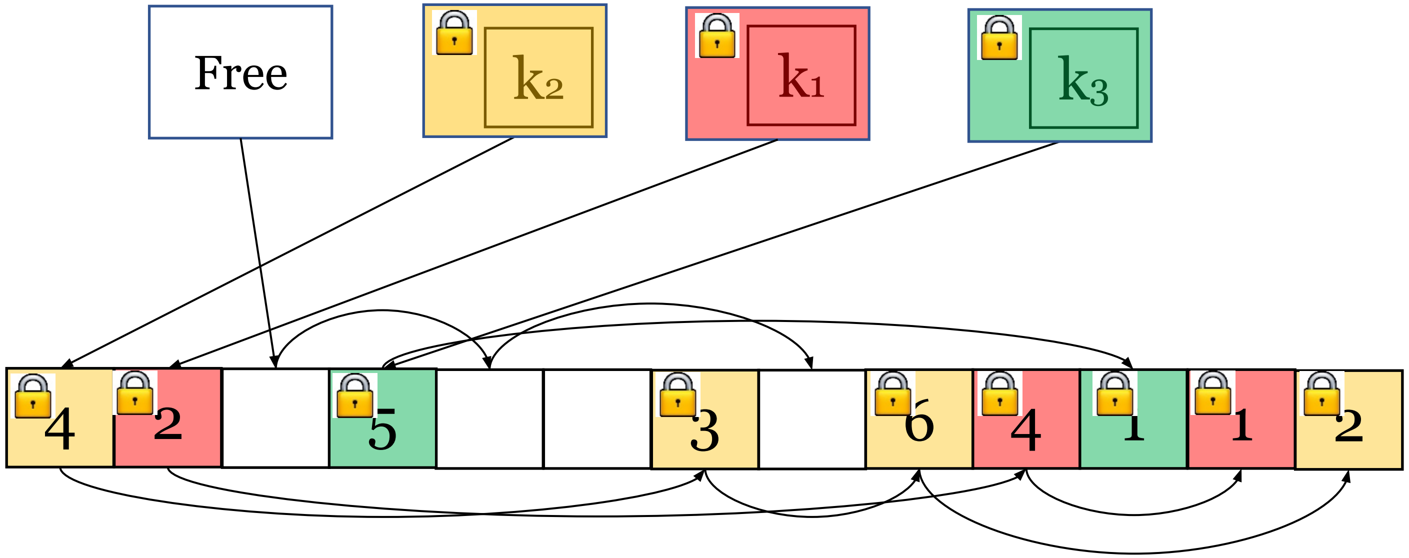
Setup 1

Let $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and $F : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$, $G : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$, and $P : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ be pseudo-random functions. Let $H_1 : \{0,1\}^* \to \{0,1\}^*$ and $H_2 : \{0,1\}^* \to \{0,1\}^*$ be random oracles. Let $z \in \mathbb{N}$ be the initial size of the free list.

- $\mathsf{Gen}(1^k)$: sample three k-bit strings K_1, K_2, K_3 uniformly at random and generate $K_4 \leftarrow \mathsf{SSE.Gen}(1^k)$. Output $K = (K_1, K_2, K_3, K_4)$.

- $\mathsf{Enc}(K, \mathbf{f})$:

  1. let $\mathtt{A}_s$ and $\mathtt{A}_d$ be arrays of size $|\mathbf{c}|/8 + z$ and let $\mathtt{T}_s$ and $\mathtt{T}_d$ be dictionary of size $\#W$ and $\#\mathbf{f}$, respectively. We assume $\mathbf{0}$ is a $(\log \#\mathtt{A}_s)$-length string of 0's and that free is a word not in W.
  2. for each word $w \in W$,[a]
     (a) create a list $\mathtt{L}_w$ of $\#\mathbf{f}_w$ nodes $(\mathtt{N}_1, \ldots, \mathtt{N}_{\#\mathbf{f}_w})$ stored at random locations in the search array $\mathtt{A}_s$ and defined as:
     $$\mathtt{N}_i := (\langle \mathsf{id}_i, \mathsf{addr}_s(\mathtt{N}_{i+1}) \rangle \oplus H_1(K_w, r_i), r_i)$$
     where id_i is the ID of the ith file in $\mathbf{f}_w$, r_i is a k-bit string generated uniformly at random, $K_w := P_{K_3}(w)$ and $\mathsf{addr}_s(\mathtt{N}_{\#\mathbf{f}_w+1}) = \mathbf{0}$
     (b) store a pointer to the first node of $\mathtt{L}_w$ in the search table by setting
     $$\mathtt{T}_s[F_{K_1}(w)] := \langle \mathsf{addr}_s(\mathtt{N}_1), \mathsf{addr}_d(\mathtt{N}_1^\star) \rangle \oplus G_{K_2}(w),$$
     where $\mathtt{N}^\star$ is the dual of $\mathtt{N}$, i.e., the node in $\mathtt{A}_d$ whose fourth entry points to $\mathtt{N}_1$ in $\mathtt{A}_s$.


Free
k_2
k_1
k_3
4
2
5
3
6
4
1
1
2

Setup 2

3. for each file f in $\mathbf{f}$,

   (a) create a list $\mathtt{L}_f$ of $\#\bar{f}$ dual nodes $(\mathtt{D}_1, \ldots, \mathtt{D}_{\#\bar{f}})$ stored at random locations in the deletion array $\mathtt{A}_d$ and defined as follows: each entry $\mathtt{D}_i$ is associated with a word w, and hence a node $\mathtt{N}$ in $\mathtt{L}_w$. Let $\mathtt{N}_{+1}$ be the node following $\mathtt{N}$ in $\mathtt{L}_w$, and $\mathtt{N}_{-1}$ the node previous to $\mathtt{N}$ in $\mathtt{L}_w$. Then, define $\mathtt{D}_i$ as follows:

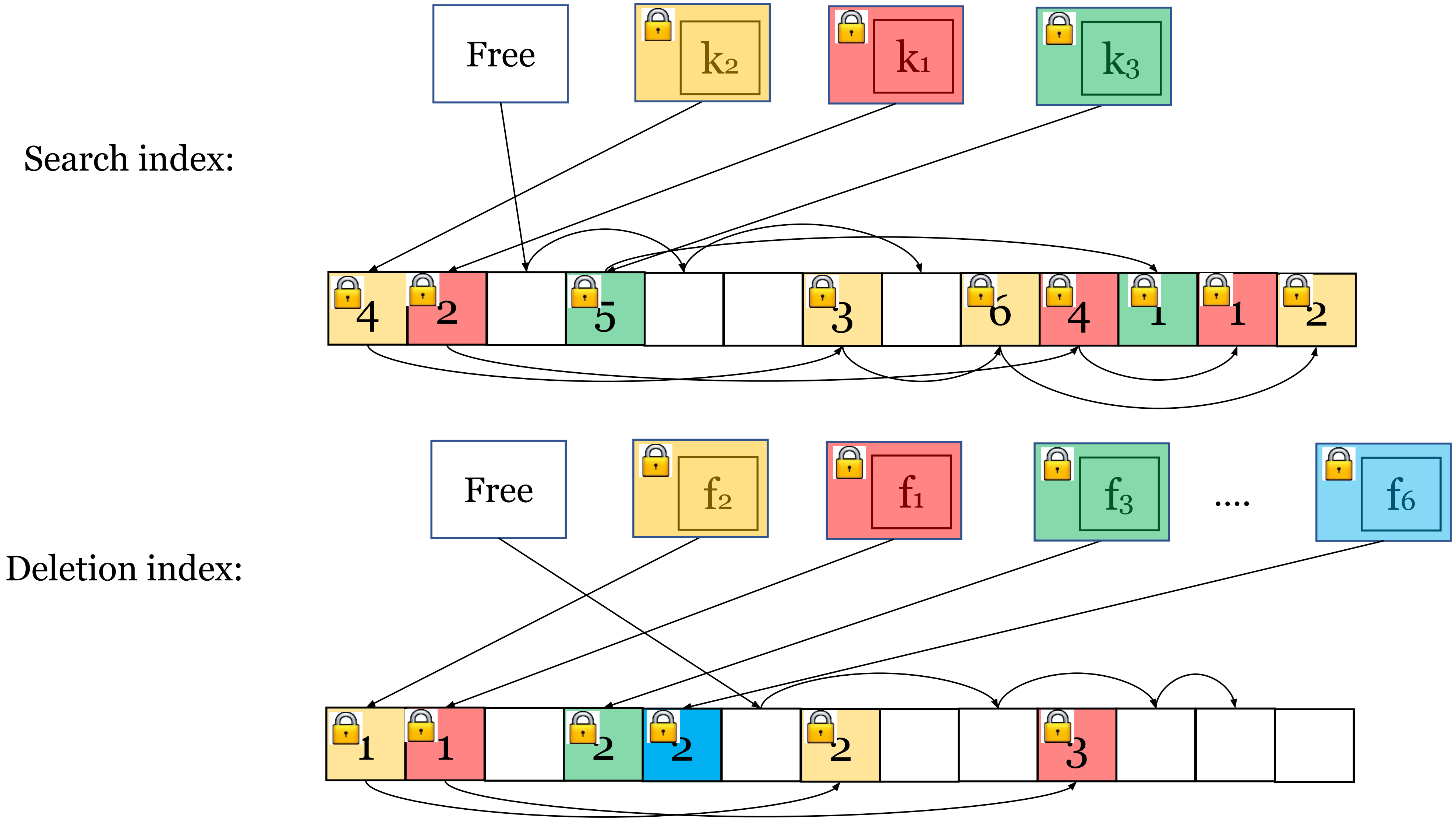
$$\mathtt{D}_i := \left(\left\langle \mathsf{addr}_d(\mathtt{D}_{i+1}), \mathsf{addr}_d(\mathtt{N}^\star_{-1}), \mathsf{addr}_d(\mathtt{N}^\star_{+1}), \mathsf{addr}_s(\mathtt{N}), \mathsf{addr}_s(\mathtt{N}_{-1}), \mathsf{addr}_s(\mathtt{N}_{+1}), F_{K_1}(w) \right\rangle \oplus H_2(K_f, r'_i), r'_i\right)$$

   where r'_i is a k-bit string generated uniformly at random, $K_f := P_{K_3}(f)$, and $\mathsf{addr}_d(\mathtt{D}_{\#\bar{f}+1}) = \mathbf{0}$.

   (b) store a pointer to the first node of $\mathtt{L}_f$ in the deletion table by setting:

$$\mathtt{T}_d[F_{K_1}(f)] := \mathsf{addr}_d(\mathtt{D}_1) \oplus G_{K_2}(f)$$

Dual Index for Deletion



Setup 3

4. create an unencrypted free list $\mathsf{L}_{\mathsf{free}}$ by choosing z unused cells at random in $\mathtt{A}_s$ and in $\mathtt{A}_d$. Let $(\mathtt{F}_1, \ldots, \mathtt{F}_z)$ and $(\mathtt{F}'_1, \ldots, \mathtt{F}'_z)$ be the free nodes in $\mathtt{A}_s$ and $\mathtt{A}_d$, respectively. Set

$$\mathtt{T}_s[\mathsf{free}] := \langle \mathsf{addr}_s(\mathtt{F}_z), \mathbf{0}^{\log \#\mathtt{A}} \rangle$$

   and for $z \geq i \geq 1$, set

$$\mathtt{A}_s[\mathsf{addr}_s(\mathtt{F}_i)] := \mathbf{0}^{\log \#\mathbf{f}}, \mathsf{addr}_s(\mathtt{F}_{i-1}), \mathsf{addr}_d(\mathtt{F}'_i)$$

   where $\mathsf{addr}_s(\mathtt{F}_0) = \mathbf{0}^{\log \#\mathtt{A}}$.
5. fill the remaining entries of $\mathtt{A}_s$ and $\mathtt{A}_d$ with random strings
6. for $1 \leq i \leq \#\mathbf{f}$, let $c_i \leftarrow \mathsf{SKE.Enc}_{K_4}(f_i)$
7. output $(\gamma, \mathbf{c})$, where $\gamma := (\mathtt{A}_s, \mathtt{T}_s, \mathtt{A}_d, \mathtt{T}_d)$ and $\mathbf{c} = (c_1, \ldots, c_{\#\mathbf{f}})$.

Search

- $\mathsf{SrchToken}(K, w)$: compute and output $\tau_s := \big(F_{K_1}(w), G_{K_2}(w), P_{K_3}(w)\big)$
- $\mathsf{Search}(\gamma, \mathbf{c}, \tau_s)$:

  1. parse τ_s as (τ_1, τ_2, τ_3) and return an empty list if τ_1 is not present in $\mathtt{T}_s$.
  2. recover a pointer to the first node of the list by computing $(\alpha_1, \alpha_1') := \mathtt{T}_s[\tau_1] \oplus \tau_2$
  3. look up $\mathtt{N}_1 := \mathtt{A}[\alpha_1]$ and decrypt with τ_3, i.e., parse $\mathtt{N}_1$ as (ν_1, r_1) and compute $(\mathsf{id}_1, \mathsf{addr}_s(\mathtt{N}_2)) := \nu_1 \oplus H_1(\tau_3, r_1)$
  4. for $i \geq 2$, decrypt node $\mathtt{N}_i$ as above until $\alpha_{i+1} = \mathbf{0}$
  5. let $I = \{\mathsf{id}_1, \ldots, \mathsf{id}_m\}$ be the file identifiers revealed in the previous steps and output $\{c_i\}_{i \in I}$, i.e., the encryptions of the files whose identifiers were revealed.

Setup 1

- $\mathsf{Enc}(K, \mathbf{f})$:

  1. let $\mathtt{A}_s$ and $\mathtt{A}_d$ be arrays of size $|\mathbf{c}|/8 + z$ and let $\mathtt{T}_s$ and $\mathtt{T}_d$ be dictionary of size $\#W$ and $\#\mathbf{f}$, respectively. We assume $\mathbf{0}$ is a $(\log \#\mathtt{A}_s)$-length string of 0's and that free is a word not in W.
  2. for each word $w \in W$,[a]
     (a) create a list $\mathtt{L}_w$ of $\#\mathbf{f}_w$ nodes $(\mathtt{N}_1, \ldots, \mathtt{N}_{\#\mathbf{f}_w})$ stored at random locations in the search array $\mathtt{A}_s$ and defined as:
     $$\mathtt{N}_i := (\langle \mathsf{id}_i, \mathsf{addr}_s(\mathtt{N}_{i+1})\rangle \oplus H_1(K_w, r_i), r_i)$$
     where id_i is the ID of the ith file in $\mathbf{f}_w$, r_i is a k-bit string generated uniformly at random, $K_w := P_{K_3}(w)$ and $\mathsf{addr}_s(\mathtt{N}_{\#\mathbf{f}_w+1}) = \mathbf{0}$
     (b) store a pointer to the first node of $\mathtt{L}_w$ in the search table by setting
     $$\mathtt{T}_s[F_{K_1}(w)] := \langle \mathsf{addr}_s(\mathtt{N}_1), \mathsf{addr}_d(\mathtt{N}_1^\star)\rangle \oplus G_{K_2}(w),$$
     where $\mathtt{N}^\star$ is the dual of $\mathtt{N}$, i.e., the node in $\mathtt{A}_d$ whose fourth entry points to $\mathtt{N}_1$ in $\mathtt{A}_s$.

Delete

- $\mathsf{DelToken}(K, f)$: output: $\tau_d := (F_{K_1}(f), G_{K_2}(f), P_{K_3}(f), \mathsf{id}(f))$.
- $\mathsf{Del}(\gamma, \mathbf{c}, \tau_d)$:

1. parse τ_d as $(\tau_1, \tau_2, \tau_3, \mathsf{id})$ and return $\perp$ if τ_1 is not in $\mathtt{T}_d$
2. find the first node of $\mathtt{L}_f$ by computing $\alpha_1' := \mathtt{T}_d[\tau_1] \oplus \tau_2$
3. for $1 \leq i \leq \#\bar{f}$,
   (a) decrypt $\mathtt{D}_i$ by computing $(\alpha_1, \ldots, \alpha_6, \mu) := \mathtt{D}_i \oplus H_2(\tau_3, r)$, where $(\mathtt{D}_i, r) := \mathtt{A}_d[\alpha_i']$
   (b) delete $\mathtt{D}_i$ by setting $\mathtt{A}_d[\alpha_i']$ to a random $(6 \log \#\mathtt{A} + k)$-bit string
   (c) find address of last free node by computing $(\varphi, \mathbf{0}^{\log \#\mathtt{A}}) := \mathtt{T}_s[\mathsf{free}]$
   (d) make the free entry in the search table point to $\mathtt{D}_i$'s dual by setting $\mathtt{T}_s[\mathsf{free}] := \langle \alpha_4, \mathbf{0}^{\log \#\mathtt{A}} \rangle$
   (e) free location of $\mathtt{D}_i$'s dual by setting $\mathtt{A}_s[\alpha_4] := (\varphi, \alpha_i')$
   (f) let $\mathtt{N}_{-1}$ be the node that precedes $\mathtt{D}_i$'s dual. Update $\mathtt{N}_{-1}$'s "next pointer" by setting: $\mathtt{A}_s[\alpha_5] := (\beta_1, \beta_2 \oplus \alpha_4 \oplus \alpha_6, r_{-1})$, where $(\beta_1, \beta_2, r_{-1}) := \mathtt{A}_s[\alpha_5]$. Also, update the pointers of $\mathtt{N}_{-1}$'s dual by setting
   $$\mathtt{A}_d[\alpha_2] := (\beta_1, \beta_2, \beta_3 \oplus \alpha_i' \oplus \alpha_3, \beta_4, \beta_5, \beta_6 \oplus \alpha_4 \oplus \alpha_6, \mu*, r_{-1}^*),$$
   where $(\beta_1, \ldots, \beta_6, \mu^*, r_{-1}^*) := \mathtt{A}_d[\alpha_2]$
   (g) let $\mathtt{N}_{+1}$ be the node that follows $\mathtt{D}_i$'s dual. Update $\mathtt{N}_{+1}$'s dual pointers by setting:
   $$\mathtt{A}_d[\alpha_3] := (\beta_1, \beta_2 \oplus \alpha_i' \oplus \alpha_2, \beta_3, \beta_4, \beta_5 \oplus \alpha_4 \oplus \alpha_5, \beta_6, \mu^*, r_{+1}^*),$$
   where $(\beta_1, \ldots, \beta_6, \mu^*, r_{+1}^*) := \mathtt{A}_d[\alpha_3]$
   (h) set $\alpha_{i+1}' := \alpha_1$
4. remove the ciphertext that corresponds to id from $\mathbf{c}$
5. remove τ_1 from $\mathtt{T}_d$

- $\mathsf{Enc}(K, \mathbf{f})$:

1. let $\mathtt{A}_s$ and $\mathtt{A}_d$ be arrays of size $|\mathbf{c}|/8 + z$ and let $\mathtt{T}_s$ and $\mathtt{T}_d$ be dictionary of size $\#W$ and $\#\mathbf{f}$, respectively. We assume $\mathbf{0}$ is a $(\log \#\mathtt{A}_s)$-length string of 0's and that free is a word not in W.
2. for each word $w \in W$,[a]
   (a) create a list $\mathtt{L}_w$ of $\#\mathbf{f}_w$ nodes $(\mathtt{N}_1, \ldots, \mathtt{N}_{\#\mathbf{f}_w})$ stored at random locations in the search array $\mathtt{A}_s$ and defined as:
   $$\mathtt{N}_i := (\langle \mathsf{id}_i, \mathsf{addr}_s(\mathtt{N}_{i+1})\rangle \oplus H_1(K_w, r_i), r_i)$$
   where id_i is the ID of the ith file in $\mathbf{f}_w$, r_i is a k-bit string generated uniformly at random, $K_w := P_{K_3}(w)$ and $\mathsf{addr}_s(\mathtt{N}_{\#\mathbf{f}_w+1}) = \mathbf{0}$
   (b) store a pointer to the first node of $\mathtt{L}_w$ in the search table by setting
   $$\mathtt{T}_s[F_{K_1}(w)] := \langle \mathsf{addr}_s(\mathtt{N}_1), \mathsf{addr}_d(\mathtt{N}_1^\star)\rangle \oplus G_{K_2}(w),$$
   where $\mathtt{N}^\star$ is the dual of $\mathtt{N}$, i.e., the node in $\mathtt{A}_d$ whose fourth entry points to $\mathtt{N}_1$ in $\mathtt{A}_s$.

3. for each file f in $\mathbf{f}$,
   (a) create a list $\mathtt{L}_f$ of $\#\bar{f}$ dual nodes $(\mathtt{D}_1, \ldots, \mathtt{D}_{\#\bar{f}})$ stored at random locations in the deletion array $\mathtt{A}_d$ and defined as follows: each entry $\mathtt{D}_i$ is associated with a word w, and hence a node $\mathtt{N}$ in $\mathtt{L}_w$. Let $\mathtt{N}_{+1}$ be the node following $\mathtt{N}$ in $\mathtt{L}_w$, and $\mathtt{N}_{-1}$ the node previous to $\mathtt{N}$ in $\mathtt{L}_w$. Then, define $\mathtt{D}_i$ as follows:

$$\mathtt{D}_i := (\langle \mathsf{addr}_d(\mathtt{D}_{i+1}), \mathsf{addr}_d(\mathtt{N}_{-1}^\star), \mathsf{addr}_d(\mathtt{N}_{+1}^\star), \mathsf{addr}_s(\mathtt{N}), \mathsf{addr}_s(\mathtt{N}_{-1}), \mathsf{addr}_s(\mathtt{N}_{+1}), F_{K_1}(w)\rangle \oplus H_2(K_f, r_i'), r_i')$$

   where r_i' is a k-bit string generated uniformly at random, $K_f := P_{K_3}(f)$, and $\mathsf{addr}_d(\mathtt{D}_{\#\bar{f}+1}) = \mathbf{0}$.
   (b) store a pointer to the first node of $\mathtt{L}_f$ in the deletion table by setting:
   $$\mathtt{T}_d[F_{K_1}(f)] := \mathsf{addr}_d(\mathtt{D}_1) \oplus G_{K_2}(f)$$

Add

- $\mathsf{AddToken}(K, f)$: let $(w_1, \ldots, w_{\#\bar{f}})$ be the *unique* words in f in their order of appearance in f. Compute

$$\tau_a := (F_{K_1}(f), G_{K_2}(f), \lambda_1, \ldots, \lambda_{\#\bar{f}}),$$

where for all $1 \leq i \leq \#\bar{f}$:

$$\lambda_i := \left(F_{K_1}(w_i), G_{K_2}(w_i), \langle \mathsf{id}(f), \mathbf{0}\rangle \oplus H_1(P_{K_3}(w_i), r_i), r_i, \langle \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, F_{K_1}(w_i)\rangle \oplus H_2(P_{K_3}(f), r_i'), r_i'\right),$$

and r_i and r_i' are random k-bit strings. Let $c_f \leftarrow \mathsf{SKE.Enc}_{K_4}(f)$ and output (τ_a, c_f).

- $\mathsf{Add}(\gamma, \mathbf{c}, \tau_a)$:
  1. parse τ_a as $(\tau_1, \tau_2, \lambda_1, \ldots, \lambda_{\#\bar{f}}, c)$ and return $\perp$ if τ_1 is not in $\mathtt{T}_d$.
  2. for $1 \leq i \leq \#\bar{f}$,
     (a) find the last free location φ in the search array and its corresponding entry $\varphi^\star$ in the deletion array by computing $(\varphi, \mathbf{0}) := \mathtt{T}_s[\mathsf{free}]$, and $(\varphi_{-1}, \varphi^\star) := \mathtt{A}_s[\varphi]$.
     (b) update the search table to point to the second to last free entry by setting $\mathtt{T}_s[\mathsf{free}] := (\varphi_{-1}, \mathbf{0})$
     (c) recover a pointer to the first node $\mathtt{N}_1$ of the list by computing $(\alpha_1, \alpha_1^\star) := \mathtt{T}_s[\lambda_i[1]] \oplus \lambda_i[2]$
     (d) store the new node at location φ and modify its forward pointer to $\mathtt{N}_1$ by setting $\mathtt{A}_s[\varphi] := \left(\lambda_i[3] \oplus \langle \mathbf{0}, \alpha_1\rangle, \lambda_i[4]\right)$
     (e) update the search table by setting $\mathtt{T}_s[\lambda_i[1]] := (\varphi, \varphi^\star) \oplus \lambda_i[2]$
     (f) update the dual of $\mathtt{N}_1$ by setting $\mathtt{A}_d[\alpha_1^\star] := \left(\mathtt{D}_1 \oplus \langle \mathbf{0}, \varphi^\star, \mathbf{0}, \mathbf{0}, \varphi, \mathbf{0}, \mathbf{0}\rangle, r\right)$, where $(\mathtt{D}_1, r) := \mathtt{A}_d[\alpha_1^\star]$
     (g) update the dual of $\mathtt{A}_s[\varphi]$ by setting $\mathtt{A}_d[\varphi^\star] := \left(\lambda_i[5] \oplus \langle \varphi^\star_{-1}, \mathbf{0}, \alpha_1^\star, \varphi, \mathbf{0}, \alpha_1, \lambda_i[1]\rangle, \lambda_i[6]\right)$,
     (h) if $i = 1$, update the deletion table by setting $\mathtt{T}_d[\tau_1] := \langle \varphi^\star, \mathbf{0}\rangle \oplus \tau_2$.
  3. update the ciphertexts by adding c to $\mathbf{c}$

Leakage

- Access pattern
- Search pattern

- Add: if keyword w appears in any other file

- Delete: pointer of previous and next element