Findex compact operation

1 Purpose of the compact operation

Findex compact operation aims to:

1. restore index compactness:

   Whenever a client adds a new associations to an existing keyword in the index, a new *link* is inserted in the corresponding keyword chain. If the previous link contained padding, the chain sparsity is increased since padding blocks exist between two non-adding blocks. The same goes for deletions since deletions consist in adding negations.

   In case most index modifications introduce padding, it becomes necessary to reduce chain sparsity to save storage. The compact operation ensures that no intra-chain padding exists once the operation is finished.

2. remove obsolete indexed data:

   Whenever an data indexed data becomes obsolete, one could want to remove all associations linking this data to a given keyword. Since no inverted index is stored by Findex for storage efficiency and security reasons, one would need to know all such associations, which is not realistic. The compact operation makes it possible to conveniently decide which data to keep in the index.

3. increase index security:

   Each SSE query leaks data. Findex is no different. While using the leaked data is difficult, it possible to deduce information on the index after the leakage reaches certain amount. One would therefore want to be able to invalid the knowledge gained by an attacker.

   The compact operation performs a cryptographic "shuffle" of the index that invalidates any previous information related to the index.

4. re-key the index:

   Whenever the Findex key becomes compromised, it is necessary to generate a new key and re-encrypt all the index.

   The Findex compact operation performs this operation by design.

2 Algorithm

- Note taken on *[2023-12-12 Tue 13:16]*
  In order to simplify the algorithm, the compact is performed on the entire tables and in one batch.

```
begin Findex.compact(k, transform):
    et-key := KDF(k, "Entry Table")
    ct-key := KDF(k, "Chain Table")

    // Rebuild the entire index.
    et-tokens := ET.list-tokens()
    et-values := ET.search(et-key, et-tokens)
    ct-tokens := #{Findex.unroll(entry) for entry in et-values.values()}
    ct-values := CT.search(ct-key, ct-tokens)

    index := {}
    for (token, entry) in et-values:
        chain-tokens := Findex.Unroll(entry)
        chain-values := [ct-values[tk] for tk in chain-tokens]
        index[entry.tag] := Findex.recompose(chain-values)
    done

    // Interactively transform the index.
    index := transform(index)

    // Write the updated index using the new key in one time.
    Findex.insert(k, index)

    // Delete the old index.
    Findex.delete({})
end
```

3 Variations 1: partial compaction

- Note taken on *[2023-12-12 Tue 15:08]*
  How to choose `f_sec`?

In practice, the index may not fit in the operator's memory, or compacting the entire index may be too long. In that case, the compact operation needs to be performed **partially**. In order to avoid reducing the security of the shuffling:

- the Entry Table is entirely re-encrypted;
- only portion of chains (chosen at random) is compacted;
- some other chains (`f_sec` times the number of compacted chains) are fetched in order to hide which chains are actually being compacted.

```
begin Findex.partial-compact(k, k', compacting-rate, filter):
    et-key := KDF(k, "Entry Table")
    ct-key := KDF(k, "Chain Table")

    // Rebuild the entire index.
    et-tokens := ET.list-tokens()
    et-values := ET.search(et-key, et-tokens)

    // Compute the number of random chains to compact so that all chains
    // are compacted at least once after 1/compacting-rate operations.
    n := collect-coupons(size(et-values), compacting-rate)

    targets := select-random(et-values, n)
    lures   := select-random(et-values \ targets, n * f_sec)

    ct-tokens := { Findex.unroll(entry) for (_, entry) in targets U lures }
    ct-values := CT.search(ct-key, ct-tokens)

    partial-index := {}
    for (token, entry) in targets:
        chain-tokens := Findex.Unroll(entry)
        chain-values := [ct-values[tk] for tk in chain-tokens]
        partial-index[entry.tag] := Findex.recompose(chain-values)
    done
```

```
    // Interactively select associations to reindex.
    partial-index := filter(partial-index)

    entries := {} // {tag:entry}
    links   := {} // {tag:link}
    for (_, entry) in targets:
        if not partial-index.contains(entry[tag])
            pass
        links = Findex.decompose(partial-index[entry[tag]])
        entry[link-counter] = size(links)
        // TODO: issue since the key is shared by all chains.
        entry[comp-counter] += 1

        entries[entry[tag]] := entry
        for i in size(links):
            ct-values'[(entry[tag] || i)] := links[i]
        done
    done

    // Insert the compacted chains
    CT.insert(ct-key', links)
    ET.insert(et-key', entries)

    // Delete the old index.
    ET.delete(et-tokens)
    CT.delete(ct-tokens)
end
```