MiniCrypt: Reconciling Encryption and Compression in Big Data Stores

Wenting Zheng, Frank Li, Rachit Agarwal, Raluca Ada Popa, Ion Stoica

**Problem**

More and more sensitive data outsourced to third parties, who may be untrustworthy

Applications need data stores that provide
- Functionality
- Data confidentiality
- Performance

Encryption → data confidentiality
Compression → better performance because more data fits in memory

**MiniCrypt**

*First system to combine compression and encryption*

Designed a KV store interface that provides functionality, data confidentiality, and performance

**Key idea:** transform key value pairs by combining small number of records into *packs*, then compress and encrypt

**System Design**

MiniCrypt is a system that enables compression and encryption, and works with existing unmodified KV stores!

An existing key value store should satisfy 2 requirements
1. Single record atomic update (UPDATE-IF)
2. Maintains sorted index on primary key

Data sorted by primary key and transformed into packs

Key value

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
<th>pack_id</th>
<th>pack value</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>encrypt + compress</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td>encrypt + compress</td>
</tr>
<tr>
<td>47</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>encrypt + compress</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td></td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Data operations**

**get(key)** Select the pack with the highest pack ID from all pack IDs that are at most key

**range(low, high)** Range query + one extra read

**put(key, value)** Each put = get + put

**Writes in MiniCrypt**

Naively implementing blind writes will cause overwrites if there are writes to different keys within the same pack

**Generic mode:**
- used in any write situation
- uses atomic update mechanism for each put

**Append mode:**
- used when data is immutable and not overwritten, and keys are increasing e.g. time-series data
- uses epoch-based logs to store new inserts; background processes merge these inserts

**Performance**

Implementation: evaluation done using Cassandra on read-only workload; we use zlib and AES encryption

*MiniCrypt provides up to 100x performance gain*

**Future Work**

- Develop more sophisticated heuristics for determining how many keys reside in a pack
- Find ways to add aggregate functionalities