Efficient Negative Databases from Cryptographic Hash Functions

George Danezis, Claudia Diaz, Sebastian Faust, Emilia Käsper, Carmela Troncoso and Bart Preneel

K.U.Leuven ESAT/COSIC

COSIC Seminar, Oct 5th 2007
Overview

- Introduction to NDBs
- Related work [Espo6]
- Properties of negative databases
- Basic idea of our schemes
- Evaluation
- Limitations of negative databases
- Conclusions
Introduction

- A negative DB is a representation of the positive DB
  - Efficient to test whether particular entries are present
  - Hard to enumerate all entries

- Applications
  - DB owners can share information
  - No-fly lists

- Contribution
  - NDB properties achievable with one-way functions
  - Improved efficiency wrt [Esp06]
  - Reducible to standard security assumptions
Related work (1)

- NDB introduced by Esponda et al. in 2004
- Complement set of the DB
  - $U$: Universe of strings of length $l$
  - $NDB = U − DB$
- String $s$ is in DB if it fails to match all the entries of NDB
- NDB represented in a compact form using wildcards ‘*’

Example:

<table>
<thead>
<tr>
<th>DB</th>
<th>U-DB</th>
<th>NDB</th>
<th>Boolean formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>001</td>
<td>$(x_1 + x_2 + \overline{x}_3)$ &amp;</td>
</tr>
<tr>
<td>100</td>
<td>010</td>
<td><em>1</em></td>
<td>$(\overline{x}_2)$</td>
</tr>
<tr>
<td>101</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Related work (2)

- **Complexity**
  - Answering “Is Q in the database?” takes time proportional to $|NDB|$, if Q is a binary string
  - Reversing NDB is $NP$-hard
  - Deciding if DB is empty is $NP$-complete
  - Answering complex queries (with arbitrary number of ‘*’ is intractable)

- **But...**
  - Some instances of NDBs may be easy to reverse
Related work (3)

- Algorithm to generate hard-to-reverse NDBs [Espo06]
  - Generate an NDB for each string in DB
  - Inexact representation of U-DB (some strings in addition to DB will not be matched by NDB) – add extra bits for validity checking
  - The length of the strings must be > 1,000 bits
  - Security evaluation: not solved after 24h with state-of-the-art SAT solvers

- Multi-record NDBs?
  - Have an NDB for each entry in DB
Properties of negative databases [Esp06]

- Hard to reverse
  - Only queries of the form “is s in DB” are efficient
- Singleton NDB
  - Negative representation of single string or no string at all
- Easy to update
  - Efficient algorithms for adding and deleting entries in DB
- Obfuscated size
  - The size of DB should not be visible from NDB
- Probabilistic
  - A string $s \in DB$ should have many possible representations in NDB
- String based
  - “Manipulating entries in NDB can meaningfully affect DB”
Our schemes

- Based on one-way functions
  - Hash functions
  - Discrete log assumption
- DB has $m$ records with $n$ fields each: $DB_{i,j} \in M$
- For each $DB_{i,j}$ we generate a random $r_{i,j} \in R$
- One-way function $H : R \times M \rightarrow T$
  - $t_{i,j} = H(r_{i,j}, DB_{i,j})$
  - $NDB_{i,j} = (r_{i,j}, t_{i,j})$
- Add $d$ dummy entries to obfuscate size
- Querying element $s$ in field $k$
  - Apply $H$ to all pairs $(r_{i,k}, s)$, check if result matches $t_{i,k}$
Back to the properties

- Hard to reverse
  - Given by the properties of one-way functions
- Singleton NDB
  - Entries in NDB correspond to an entry in DB or to a dummy
  - Negligible error probability
- Easy to update
  - Apply H to entry and add to (or remove from) NDB
- Obfuscated size
  - Dummy and real entries indistinguishable (upper bound on size)
- Probabilistic
  - Random values of $r_{i,j}$, one chosen at random
  - Hard to determine that two entries represent the same value
- String based: Not provided by our schemes
  - We have not found any functionality provided by this feature that our schemes cannot satisfy
Evaluation

- We compare the efficiency of our approach wrt Espo6 in terms of space and time complexity

<table>
<thead>
<tr>
<th></th>
<th>Espo6</th>
<th>Our Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Esopo6</td>
<td>• Each entry $s$ in DB is represented by a whole NDB of size $O(l^2)$ bits ($l&gt;1,000$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Total size of negative image is $O(m \cdot l^2)$</td>
<td>• Each entry $s$ in DB is represented by an entry in NDB of size $O(t+r)$ bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Total size of negative image is $O(m \cdot (t+r))$</td>
</tr>
<tr>
<td><strong>Query time</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Esopo6</td>
<td>• Need to check $m \cdot l$ entries</td>
<td>• Need to check $m$ entries, $t_H$ time to execute the one-way function</td>
</tr>
<tr>
<td></td>
<td>• Query time is $O(m \cdot l)$</td>
<td>• Query time is $O(m \cdot t_H)$</td>
</tr>
</tbody>
</table>
Experimental results

Creating a NDB

Query response time

[1 GHz Pentium]
Limitations

- Entropy of the data
  - Date of birth
  - Frequent names/surnames
- Aggregate several low-entropy fields together
  - Disables searches on specific fields
- Add high-entropy keys specific to the individual
  - E.g., passport number
Conclusions

- Practical and efficient scheme for implementing NDBs
- Security reducible to well understood cryptographic assumptions
  - As opposed to relying on instances of SAT formulas that seem intractable for SAT solvers
- Cost is $O(m \cdot l)$ space and $O(m)$ time
  - NDB could be smaller than DB for large $l$
  - Multiple fields increase cost (time and storage) linearly
  - As opposed to $O(m \cdot l^2)$ space and $O(m \cdot l)$ time
- Future work
  - Proving statements about entries in Zero-Knowledge
Thank you!