# CRYPTOGRAPHIC PROTECTION OF MEMBERSHIP LISTS*

JOAN FEIGENBAUM†, ERIC GROSSE† , AND JAMES A. REEDS†

**Abstract.** We present a simple algorithm for encrypting membership lists. The encrypted lists and search software can be distributed to members of the organization, who can use them to make legitimate queries. However, it is difficult to use the encrypted list for unauthorized purposes, such as the generation of sales lists. We illustrate the method by applying it to the SIAM membership list, yielding a package that can be widely distributed.

**1. Introduction.** Professional societies have long faced the question of what to do with their membership lists. There are clearly good reasons to give copies of the list to all members of the society: Communication among the members serves the goals of the profession and fosters a sense of camaraderie among the members. On the other hand, if numerous copies of the membership list are in circulation, some are bound to fall into the hands of insurance salesman, senders of junk mail, rival societies, etc. This dilemma is, of course, faced by many organizations besides professional societies and is exacerbated by the existence of computerized lists.

We propose to ameliorate the problem by cryptography. Given a membership list $L$, our method produces an encrypted list $L'$ and a lookup program $P'$ with the following properties.

- For any query key $q$, $P'(q, L') = \{\, v \in L \mid q \,\mathrm{matches}\, v \,\}$.
- Given $L'$ and $P'$, it is difficult to reconstruct $L$.

This is a special case of the *database protection problem.* That is, the databases in question are simply a collection of records with explicit search keys. A more complicated special case, that of automaton-structured linguistic databases, is considered in [FLW89].

**2. Algorithm.** The starting point for our algorithm is a standard table-encryption technique first developed by Needham (see Denning's book [D82, Section 3.6] for a thorough discussion).

Let $L$ be a set of (*key,value*) pairs $\{\, (k_i, v_i) \mid 1 \le i \le n \,\}$ where $k_i$ come from some set $K$ of syntactically valid keys. Construct a corresponding $L'$ and an access program $P'$ as follows.

Choose a one-way hash function $h$ whose domain is $K$. Ideally, such a function $h$ would be polynomial-time computable, mapping from $K$ into integers 1 through $M$ with the property that, given $1 \le j \le M$, it is computationally infeasible to find $h^{-1}(j) \subset K$. For a more thorough discussion of one-way hash functions, refer to [M89, NY89].

Let $Enc$ and $Dec$ be encryption and decryption functions such that for all $x$ and $y$, $Dec(Enc(x, y), y) = x$. An encrypted list $L'$ is derived from $L$ as follows: For every $(k, v)$ pair in $L$, a pair $(h(k), Enc(v, k))$ is inserted into $L'$. The lookup program $P'$ can be as simple as

* Typeset on December 11, 1991.

† AT&T Bell Laboratories, Murray Hill NJ 07974 USA.

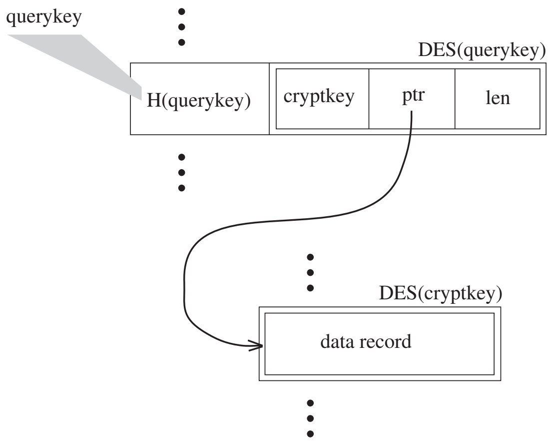


FIG. 1. *To allow searching by multiple keys, we use an indirect form of the usual encrypted database structure. Interpolation search using a hashed query key yields, after one deciphering step, a crypt key, pointer, and length. These are used, with a second deciphering step, to retrieve the data record.*

```
P'(k, L')
    {
        k' ← h(k)
        for all v' such that (k',v')∈ L'
            output(Dec(v',k))
    }
```

Since the hashed keys are uniformly distributed, the **for** loop only requires a (very fast) interpolation search.

In our application, the values $v_1$, …, $v_n$ are the membership records. That is, $v_i$ contains the name, address, phone number, etc. for the $i^{th}$ member. The standard table-encryption technique assumes that the key-value relation is one-to-one. This is a problem because in our case, we wish to retrieve a record by the last name, which may correspond to multiple records. More importantly, we may wish to allow access to records by other keys as well.

Consequently we insert a layer of indirection. Let $v_1, \ldots, v_n$ be a list of membership records and $S_1, \ldots, S_n$ be the corresponding sets of keys. For example, we later choose $S_i = \{$last name, phone number$\}$ of the member in record $v_i$. Let $r_1$, …, $r_n$ be a list of random numbers. The encrypted membership list contains records $E_1 = Enc(v_1, r_1)$, …, $E_n = Enc(v_n, r_n)$. These variable length records are stored in a block of memory that we will refer to as *MainTable*.

Let $K_{in} = \{k_1, \ldots, k_m\} = \bigcup_{i=1}^{n} S_i$. A second block of memory, referred to as *AuxTable*, contains the fixed length records $(h(k), w = Enc(\Lambda, k))$ sorted in increasing order by $h(k)$ values. Each $\Lambda$ is a list of triples of the form $(r, a, l)$: $r$ is the random number associated with a membership record $v$ that has key $k$; $a$ is the address of $v$ in *MainTable*; $l$ is the length of record $v$. So if $k$ is the last name Cohen, $\Lambda$ is a list of the random numbers, addresses, and lengths of all records of members named Cohen. The contents of *AuxTable* and *MainTable* are summarized in Figure 1.

The lookup program $P'$ becomes

```
P'(k, AuxTable, MainTable)
    {
        k' ← h(k)
        if k' ∉ AuxTable, return(∅)
        j ← index of k' in AuxTable
        Λ ← Dec(w[j], k)
        for (r, a, l) ∈ Λ
        {
                v' ← MainTable[a, a+l-1]
                output(Dec(v',r))
        }
    }
```

There is a small cheat here. The hash function $h$ is in general not one-to-one, and hence $k$ may not be the correct decryption key for $w_j$. A practical implementation therefore checks that $a$ points inside *MainTable* and that $l$ is not too large; if it is paranoid, it also verifies that $k$ really is a key for each decrypted record $v$.

An organization can distribute the lookup program $P'$ along with the membership information stored in *AuxTable* and *MainTable*. Because the membership records are distributed in encrypted form, they cannot be used directly for illegitimate purposes such as the generation of sales lists. Of course, an attacker could produce a complete set of cleartext records by exhaustively searching the keyspace. This weakness is unavoidable if the lookup program is to meet its specifications. In this application, the level of security appears to be appropriate to the level of the threat.

**3. Example: The SIAM Membership List.** By special arrangement, SIAM periodically sends netlib[DG87] a dump of its member database. Currently there are 8176 records, with an average size of 95 bytes (name, address, phone number, and email address) for a total of 777 kilobytes. The most common last name, Chan, matches 44 records. Note that one copy of the list fits on a floppy disk, but two copies (as would be needed by naive table encryption) do not fit.

A practical realization of $P'$ requires the choice of the hashing and encryption functions $h$, *Enc*, *Dec*. Since we wish to abide by both the letter and the spirit of stringent United States export laws, this choice was in fact the biggest hurdle of the project. (Export is important because SIAM is an international organization and, besides, netlib has no way to verify the destination country of email.) Ultimately, we chose the Data Encryption Standard (DES) [D82, M89] for all three functions. Many Unix$^{TM}$ systems are delivered with library functions for this, so our program uses the library and does not include DES code. Fortunately, a DES implementation by Stig Ostholm of Finland is publicly available by anonymous ftp from `chalmers.de` as file `ftp/pub/des`. The lookup program we distribute only uses DES in one direction, so there is no point trying to use it for encrypted communication.

In the United States, a ten digit telephone numbering scheme is used that currently allows about 10,000 valid six digit prefixes. To exhaustively try all phone numbers therefore takes roughly $10^8$ tries. Let us postulate an attacker armed with a modern workstation and DES implementation that executes in ten microseconds. Decrypting one record is no help in cracking others, and our attacker's objective is the entire list. If the cost of the hash function were just one DES call, the $10^8$ tries could be done in a few thousand seconds. That is unacceptable, so we use multiple rounds of DES. There is a delicate balance here between making lookup slow enough to discourage crackers with fast workstations, yet fast enough to allow quick response on a slow personal computer. At the present state of technology, about 30 rounds of DES per cipher operation is a reasonable tradeoff. (Twenty cpu days and a little programming skill would be needed on the hypothetical workstation.) Note that the multiple rounds of DES are only needed for hashing and for encrypting the pointer, not for encrypting the actual data. The distributed program reads the number of rounds from the database, to allow transparent change in later distributions of the database as computers get faster.

Another possible attack would be to try a list of common names. A list of 1095 most common last names from a Portland telephone book matched 1925 out of the total 8176 records in the SIAM example; a larger list of 18031 names from Bell Laboratories matched 3964 records. Even easier in this particular application would be to skip the encrypted database altogether and key or scan in the Combined Membership List published by the mathematical societies. Ultimately, we have to rely on the copyright notice attached to the database and the express prohibition against misuse. The purpose of the technical scheme is to dissuade casual attackers.

For details of implementation, the reader is invited to examine the code, which is available by

```
mail netlib@research.att.com
send decryptdb.c from research
send siamdb.uu from research.
```

or, if email is inconvenient, by

```
ftp research.att.com
Name:  netlib
Password:  <your email address>
binary
cd research
get decryptdb.c
get siamdb.uu
```

This version of the database is a small sample from the start of the SIAM membership list. Since the encrypted file is binary, for email purposes it is shipped in a form that must be unpacked using the `uudecode` Unix system command. (After the necessary approval by SIAM, the full database will be installed.) Compile the C program; then to search for, say, Dr. Aavatsmark's entry

```
decryptdb siamdb Aavatsmark
```

(Case is significant, so be careful to capitalize the name.)

The code is mostly a straightforward implementation of Figure 1. One subtlety is that decryption of the data record actually uses the "encrypt" direction of the `encrypt` library function. This is because some vendors provide a library that does not allow the "decrypt" direction.

The alternative to distributing membership lists is to provide remote access to a central database. The SIAM membership list has long been available in this way by sending a `whois` message by email to netlib. The AMS has recently begun to provide access to the mathematical societies' Combined Membership List by interactive access over the Internet. These are popular services and should continue. One difficulty these alternatives do have is the startup delay. We dream of a day when your telephone, which already can tell you the calling number, will conspire with your local computer to tell you who belongs to that number, even if you've never had a call from him before.

*We thank Matt Bishop and Doug McIlroy for helpful discussions.*

## REFERENCES

[D82] D. Denning. *Cryptography and Data Security*, Addison-Wesley, Reading, 1982.

[DG87] Jack J. Dongarra and Eric Grosse. Distribution of mathematical software via electronic mail. *Communications of the ACM* (1987) 30:403–407.

[FLW89] J. Feigenbaum, M. Liberman, and R. Wright. Cryptographic Protection of Databases and Software, *Proceedings of the DIMACS Workshop on Distributed Computing and Cryptography* (Princeton, NJ; October, 1989), AMS/ACM, 1991, 161-172.

[M89] R. Merkle. One-Way Hash Functions and DES, *Proceedings of the 9th CRYPTO* (1989), Springer-Verlag LNCS 435, 428–436.

[NY89] M. Naor and M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications, *Proceedings of the 21st STOC* (1989), ACM, 33–43.