

Semantic Encoding of Relational Databases in Wireless Networks

D. Paul Benjamin ^{*a}, Adrian Walker ^{**b}

^aComputer Science Dept., Pace University, 1 Pace Plaza, New York, NY USA 10038

^bReengineering LLC, P.O. Box 1412, Bristol CT, USA 06011

ABSTRACT

Semantic Encoding is a new, patented technology that greatly increases the speed of transmission of distributed databases over networks, especially over ad hoc wireless networks, while providing a novel method of data security. It reduces bandwidth consumption and storage requirements, while speeding up query processing, encryption and computation of digital signatures. We describe the application of Semantic Encoding in a wireless setting and provide an example of its operation in which a compression of 290:1 would be achieved.

Keywords: database compression, database security, wireless networks, distributed databases

1. INTRODUCTION

The current trend in databases is mobile, wireless applications. Both consumer and military database applications are moving from traditional wired networks to wireless ones to meet the demand for access in any place at any time. The size of wireless networks is growing rapidly, as is the size of the databases distributed across them, giving rise to new security and consistency problems. These problems will only get worse, as future increases in requirements always outstrip increases in bandwidth.

In contrast to the traditional wired setting, wireless database users are infrequently connected to the network and stay connected only for short periods of time. This restricts the volume of data transfer, which is already restricted by the bandwidth limitations of wireless networks, and causes data and applications to become much more location dependent.

The goal of traditional software design is to make the data and applications appear location-transparent to the user, i.e. no matter where the user is the data appear the same and the applications function the same. But for large wireless networks this cannot remain true. Disconnected users will retain versions of the database that are inconsistent with other nodes' versions, and transactions made at a node may conflict with transactions made at other nodes, complicating the process of committing transactions and requiring resources to be expended in checking consistency and rolling back transactions. This problem is exacerbated by the ever-increasing size of databases, which results in a smaller fraction of the database residing at each node. The result is that mobile, wireless databases tend to be represented as small, inconsistent fragments spread across the network. This is also true for the network's own routing tables. For networks of a few hundred nodes, various techniques have been developed for handling inconsistency, and they appear to work well [3], especially when multiple users tend not to update the same parts of the database. But these techniques do not scale successfully to very large networks [5, 6].

* benjamin@pace.edu

** adrianw@snet.net

Another result of the impermanent connectivity of the network is that data may not be available when a node needs it. This is typically addressed by redundant storage of the data at multiple nodes, increasing the likelihood that a node with the requested data will be available. Redundant storage also reduces the database's vulnerability to malicious tampering. But the resulting increase in bandwidth consumption and storage space can be excessive.

In addition, wireless users are not easily identifiable by their connections, as is the case in wired networks. Anyone with similar hardware can eavesdrop on the signal and make a copy of the data. Anyone can spoof a node and pretend to be a legitimate server and thereby provide incorrect data to clients. This requires a client to be able to verify the identity of the server.

In summary, mobile databases in a wireless network are faced with serious security and consistency issues. Each "hop" from one node to another must be protected so that the sender is clearly identifiable and the data is available only to the intended recipient, and the inconsistency that results from fragmentation of the database must be reduced.

2. COMPRESSING AND PROTECTING DATABASES

Numerous tools have been developed to address the problems outlined above. Most of these are designed for wired networks, although a few are specifically tailored to wireless networks. All of these tools use some combination of data compression and encryption.

It is universally recognized that databases should be compressed. The growth in the size of databases always outstrips the growth in storage capacity and bandwidth. Large databases are currently in the multi-terabyte range, e.g. the IBM patent database is 15 Terabytes uncompressed. Soon, large databases will approach petabytes in size.

Databases are typically compressed using well-known syntactic compression methods such as LZW. These syntactic data compression methods work by analyzing the frequency of occurrence of bit patterns in the text and re-encoding frequent patterns with shorter bit patterns. Typically, compressing a database in this fashion can produce a compression factor between 2 and 4, i.e. the compressed database is one-half to one-quarter its original size [7].

Compression results in a savings in transmission time and storage space, and also has an unexpected benefit: *query processing is actually faster on a compressed database* than on the uncompressed database [7, 8]. The savings in transmission time results in a reduction in inconsistency, because local versions of the database are transmitted farther, reducing the number of distinct versions of the database in the network. Because of these benefits, compression is widespread in database applications. Another benefit of compression is that security algorithms such as encryption and message hashing can run faster on a compressed database, e.g. a database compressed by a factor of 4 can take one-fourth the time to encrypt and decrypt, as long as the compression algorithm and the encryption algorithm are independent.

Once a database has been encrypted with a symmetric key, the symmetric key can be encrypted with public key technology. Using the intended recipient's public key for this purpose guarantees that only the intended recipient will be able to decrypt the database.

Similarly, a digital signature can be used to identify the sender of a database and verify that it has not been tampered with. This is accomplished by generating a message digest for the database, using a secure hashing algorithm. Then this digest is encrypted using the sender's private key.

In principle, this combination of compression and encryption solves the security problems associated with mobile databases in a wireless environment, and compression reduces the inconsistency. But this approach is very expensive. The database will be replicated at many nodes of the network, consuming a great deal of storage space. Similarly, the bandwidth consumed by the transmission of multiple copies will slow down other communications on the network. And the computation time required at the nodes can be significant, e.g. for computing a message digest of a large database [2]. In practice the computing power available at wireless nodes consists of handheld devices and laptop

computers that are relatively slow on these computations and will remain so for the near future. The compression factor is a critical factor in reducing the cost of storage, security computations and transmission time.

Syntactic compression methods have another drawback: it is generally not useful to apply syntactic compression to an encrypted database. This is because encryption changes the binary encoding so that frequency patterns are eliminated. Thus, compression must be performed before encryption. But it is frequently the case that compression methods are built-in to transmission software and hardware, compressing data immediately before it is transmitted. Such methods have no effect on encrypted databases. And in many applications a database is shared by multiple users who keep their records encrypted. Therefore, it is necessary for compression to be part of the database software itself, so that it can be performed before encryption.

We have developed Semantic Encoding specifically to address these issues, by providing an *integrated* mechanism of compression and protection that greatly reduces these costs. Semantic Encoding is a new technology for protecting and compressing relational databases that is especially useful for wireless mobile applications, because it:

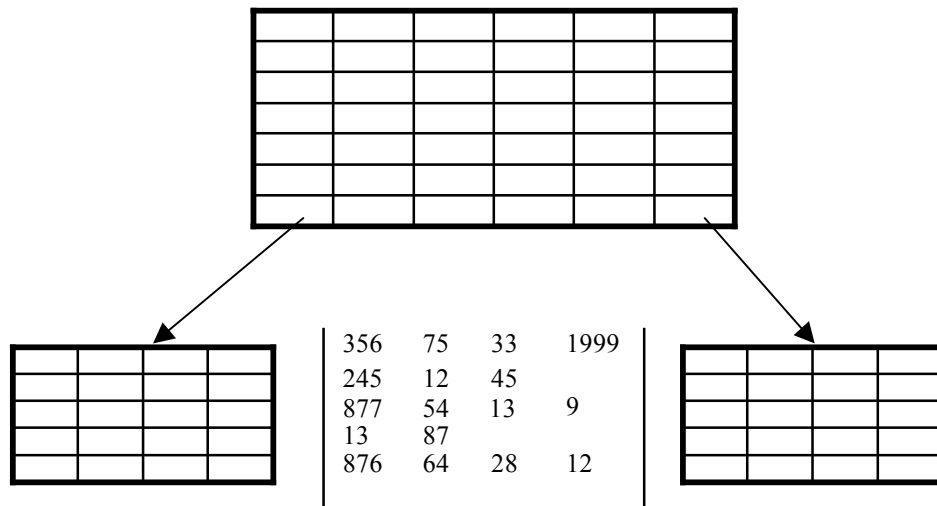
- Increases the database's compression factor, and
- Encrypts the database in a novel fashion that is significantly faster
- The encrypted database is immune to conventional cryptological attacks.

3. SEMANTIC ENCODING

Semantic Encoding is an approach to database compression and protection that is fundamentally different from previous approaches. Existing compression techniques compress text by recoding frequently occurring letters with shorter codes, and existing data protection methods work by transforming data into a string of bits that appears to be random. Semantic Encoding simultaneously compresses and protects a relational table by transforming it into a collection of components that can be reassembled in an extremely large number of meaningful ways. This transformation is accomplished by *projecting* vertical slices of the table into smaller tables and constructing a compact, secure interconnection matrix. The table is stored in this projected form. Projection can compress the table by a factor much larger than can be achieved by syntactic compression methods, typically compressing the original table by a factor of ten, a hundred or more.

The compression and security achieved by Semantic Encoding result from the fact that when a table is projected, many repeated entries are eliminated. For example, if a bank database containing information about depositors' transactions is projected, one of the subtables would be a table of all the depositors. Each depositor will occur many times in this projected table, once for each transaction. The repeated occurrences of each depositor are removed from the projected table, so that each depositor occurs only once. This compresses the table and also protects it, because the problem of reassembling two projected subtables to obtain the original table is known to have no solution [9].

The figure on the next page shows a table projected into two subtables together with an *interconnection matrix*. Each row in the matrix specifies the entries in the right-hand table that should be joined with the corresponding entry in the left-hand table, e.g. the first entry in the left-hand table should be joined with entries, 356, 75, 33 and 1999 in the right-hand table to produce the correct entries in the original table. This information is essential to be able to reassemble the subtables.



Once the table has been projected and an interconnection table produced, Semantic Encoding creates a *permutation* for that matrix, which changes the order of the rows of the matrix. The permutation is typically generated from an encryption key, but can be generated by other sources such as entropy generators. The permutation is the key for reassembly of the table. Permuting the table protects it in a unique way: an intruder without the correct permutation can reassemble the subtables in exponentially many ways that look correct but are wrong.

For example, if we apply the permutation (1 3 4 2 5) to the interconnection matrix shown above, then row 1 would be moved to the third row, row 3 would become the fourth row, row 4 would become the second row, row 2 would become the fifth row, and row 5 would become the first row. Using this scrambled interconnection matrix to reassemble the table would match depositors with transactions that are not theirs.

There are 120 distinct permutations of the rows of this small matrix. This number grows exponentially as the number of rows in the table. For a real database, the number of possible permutations is extraordinarily large. Even for a matrix with just 100 rows, the number of permutations is larger than 10^{150} . This means that it is completely impractical to search through all the permutations. Furthermore, it would be pointless to try, because there is no way for an attacker to know when he has correctly reassembled the table. Using any permutation of the interconnection matrix will produce a table connecting real depositors to real transactions; all these reassemblies look real, but only one is correct.

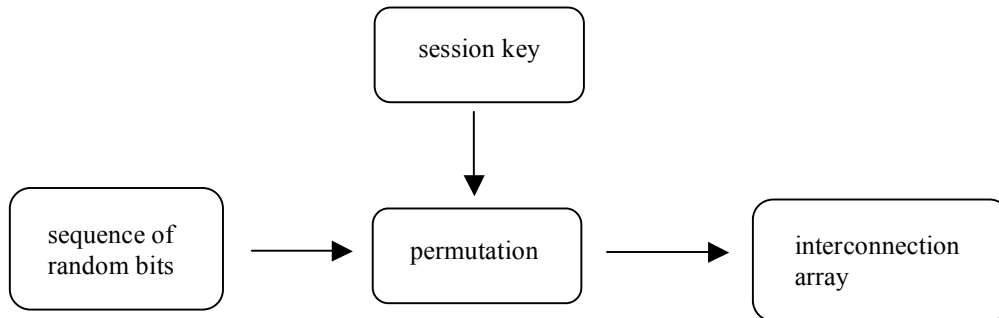
The intruder can never know if he has reassembled the projections correctly just by examining the reassembled table. In this respect, Semantic Encoding is very different from any form of encryption; when an attacker obtains an encryption key (perhaps by stealing it) he can tell if it is the correct key because it transforms the apparently random text into intelligible text. Using Semantic Encoding, all reassemblies of the subtables appear intelligible and possibly correct, but only one reassembly is actually correct.

Semantic Encoding is not a replacement for encryption; they are complementary technologies. Encryption protects the contents of each individual field in a database, and Semantic Encoding protects the relation between the fields. The combination of these two technologies provides the best security for a database.

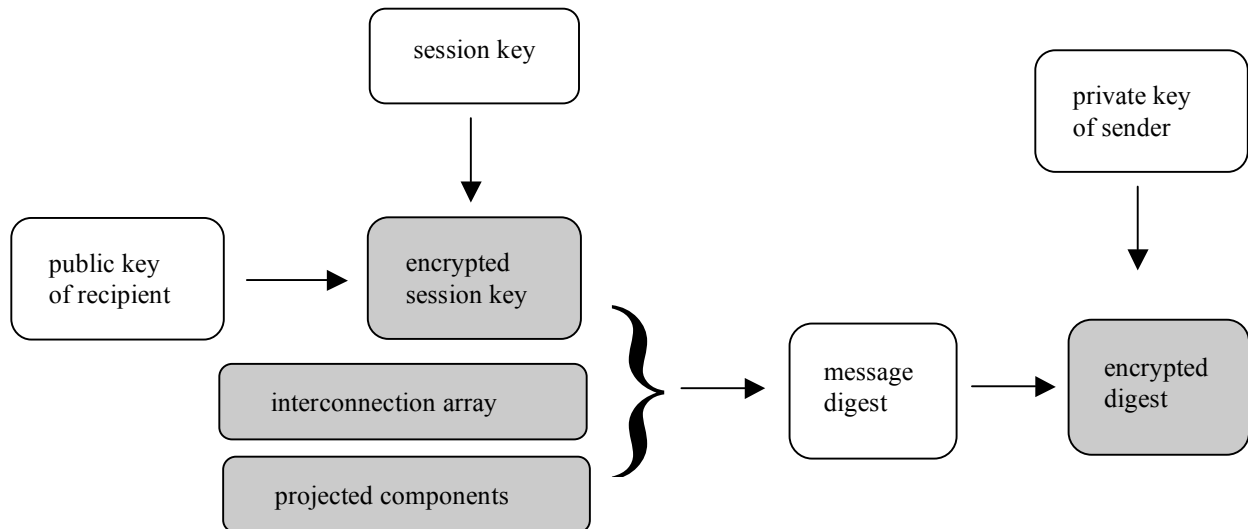
4. PROVIDING SECURITY ON THE WIRELESS NETWORK

Semantic Encoding's compression greatly reduces the time required to encrypt, digitally sign and transmit data over a wireless network. Using Semantic Encoding, a database is kept permanently in projected form. This reduces storage requirements and speeds up query response. The Semantic Encoding technology [1,10] includes algorithms for querying the encoded data, inserting new records, updating existing records, and deleting records in this projected form.

The second major contribution of Semantic Encoding is a novel security mechanism. For transmission over the wireless network, a session key is generated and used to create a permutation, using the method described in [4, pp. 229-230]. This method feeds a sequence of random bits into an encryption algorithm, producing a sequence of pseudo-random integers. This sequence is then interpreted as a permutation of the rows of the interconnection array in the straightforward way: the sequence $(i_1 \dots i_{j-1} i_j i_{j+1} \dots i_n)$ is a permutation in which row i_{j-1} is moved into row i_j , row i_j is moved into row i_{j+1} , and so forth, with row i_n moved into row i_1 .



Then the session key is encrypted with the public key of the intended recipient, so that only the intended recipient will have access to the session key. (For multiple recipients, copies of this key are made and encrypted with the public keys of each recipient.) The message to be sent consists of the projected components, the permuted interconnection array and the encrypted session key. A message digest is computed on this message, and this digest is encrypted with the sender's private key.



The message is transmitted to the recipient (including the encrypted key), together with the message digest. The gray boxes in the above diagram indicate the information that is transmitted. Everyone who receives this message can verify the identity of the sender, but only the proper recipient can decode the session key and unscramble the database properly.

This scheme provides a fast and reliable means for the recipient of a message to identify the sender, and for the sender to ensure that only the intended recipient can use the data.

Encoding the database in this way significantly reduces costs. The storage space required is reduced by the compression factor. If the projected tables are encrypted with the session key (optional) then this is speeded up by the same factor. Scrambling of the interconnection array using the permutation is faster than encrypting it, and by the same factor. Computing a message digest is similarly faster. And finally, the transmission time is speeded up by the same factor, so that the overall cost of encoding and transmitting is decreased by a factor equal to the compression factor. The same analysis holds for decoding the transmission.

Thus we see that Semantic Encoding's combination of projection and permutation provides speed and security for the mobile database user by simultaneously reducing the storage requirements of every node in the network, the computation time required of every node for security-related computations, and the transmission time required to propagate the database throughout the network. Further implementation details of Semantic Encoding are presented in a separate paper [1], together with more examples.

5. AN EXAMPLE

Let us examine how Semantic Encoding can be used to protect a very large database. The numbers and attributes in this example are typical, but are not drawn from a real scenario, and are intended only to provide a concrete illustration of the encoding method and the resulting savings.

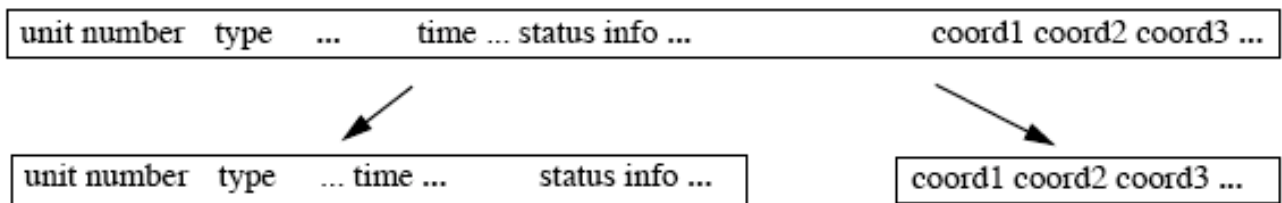
In a battlespace scenario, a commander may possess a database with records of the form:

unit number	type	configuration	...	status info	...	coord1	coord2	coord3	...
-------------	------	---------------	-----	-------------	-----	--------	--------	--------	-----

which describes each unit at a particular moment in time, its status in detail, and gives its position coordinates. Assume each record occupies 10 Kbytes, and that the database contains one hundred million records reflecting the activity of a large number of units (planes, tanks, ships, trucks, jeeps, medical units, etc.) over a period of time. The database then occupies 1 terabyte. The commander is faced with at least two major security requirements:

- to protect the data so that if the computer were to fall into enemy hands or be penetrated via a network connection then the enemy would be unable to use the information, and
- to transmit the data quickly to other commanders while ensuring that it cannot be intercepted and used by the enemy.

To accomplish both goals simultaneously, he projects the database into three pieces. First he projects the database to separate the position information from the rest:



Let us assume for the purpose of the example that the complete position relation contains 1,000,000 records (there are a million distinct positions in the battlespace) and the complete type-status relation contains 500,000 records (these are all the possible combinations of type and status). The full join of these two relations would be 500 billion records; this represents the full set of possible combinations of unit/status/position. The actual 100 million records are one-fiftieth of one percent of these possibilities. So an attacker has at best a 1 in 5000 chance of guessing which particular location corresponds to a given unit/status record in the database. As previously described, the attacker has no way of knowing whether such a guess is correct.

Let us also assume that the position records each are 3K bytes in size, and the type-status records are 7K bytes each. The projected position information table occupies 3 gigabytes, and the projected type-status information occupies 3.5 gigabytes. The size of the projected database is the size of these two tables, plus the size of the interconnection array. The interconnection array contains 500,000 rows (corresponding to every possible unit/status pair). Each row contains an array of integers specifying which rows of the position table are to be joined with it. As there are a hundred million records in the database, the average number of integers in each row is 200. Each integer takes four bytes to specify a row number in the type-status relation. Thus, the interconnection array takes 400,000,000 bytes. The total storage used for the compressed database after this first projection step is 6.9 gigabytes, so the compression factor achieved is **144:1**.

We further protect the database by applying projection again to the type-status array. Assume there are 1,000 distinct possible types in the unit relation, and 10,000 distinct possible entries in the time/status relation. Assume each type record takes 3K bytes, and each time/status record takes 4K bytes. Then the projected type relation takes 3 megabytes, and the projected time/status information takes 40 megabytes. The interconnection array has 1000 rows (one for each unit record), and the average number of entries in each row is 500 (as there are 500,000 records.) Each entry is a four-byte integer as before, so the interconnection array occupies 2 megabytes. The total size of the projected unit-status data is again the size of the two tables plus the interconnection array, which is 45 megabytes.

The total size of the entire database is now 45 megabytes plus 3 gigabytes for the position table plus 400 megabytes for the first interconnection array, which is 3.445 gigabytes. The overall compression factor is **290:1**. The result is that the transmission time required for this database (and its storage requirements) have been reduced by 99.7%

In addition to the large improvement in transmission time and storage space resulting from compression, there is a considerable speedup in security processing time from Semantic Encoding's security mechanism. For secure transmission, a session key is created and used to generate a permutation. The projected tables are encrypted with the session key for greater security. This encryption process is more than 290 times faster than encrypting the original database. It is more than 290 times faster because the interconnection arrays are not encrypted. The interconnection array is scrambled with the permutation. There are only 1000 rows in the array, and scrambling them involves changing the values of 1000 pointers from the first subtable to the interconnection array. This takes a few seconds at most. Finally, a message digest is computed so that the sender can be identified. This computation can be very time-consuming on a large database, but runs 290 times faster on the database in its projected form. Overall, the time required for security-related computations is reduced by approximately the compression factor, which for our example means a 99.7% reduction.

6. SUMMARY

Semantic Encoding is a new method of database compression and security that transforms and encodes the relation in a relational database rather than the fields. The relation is transformed by projecting it into two or more subrelations. This compresses the relation and often results in very large compression factors. The process of projection and compression is not invertible; the projected components can be reassembled in an extremely large number of ways. The transformed relation is then encoded by application of a permutation, which is typically generated from an encryption key.

The large compression factors achieved by Semantic Encoding result from a time-space tradeoff. A great deal of space is saved, at the expense of the time spent projecting and reassembling the tables. The time that is spent is regained in three ways: first, the time needed to transmit the database over a network is reduced by the factor that the database is compressed, whether the network is wired or wireless; second, query processing is faster in the compressed database; third, the time needed for encryption and computation of message digests is reduced. The overall result is a savings in both space and time, together with an additional layer of security.

Semantic Encoding is protected by U.S. Patent No. 6,691,132.

REFERENCES

1. D. Paul Benjamin and Adrian Walker, *Semantic Encoding of Information – A New Method of Data Security*, white paper, www.semanticencoding.com.
2. K.S. McCurley, *A fast portable implementation of the secure hash algorithm, III*, Technical Report SAND93-2591, Sandia National Laboratories, 1994.
3. E. Pitoura and B. K. Bhargava, *Data consistency in intermittently connected distributed systems*, Knowledge and Data Engineering, vol. 11, no. 6, pp. 896-915, 1999.
4. William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes in C*, Cambridge University Press. 1988.
5. D. Ratner, P. Reiher, and G. Popek, *Roam: A Scalable Replication System for Mobile Computing*, In Workshop on Mobile Databases and Distributed Systems (MDDS), September 1999.
6. David Ratner, Peter Reiher, Gerald Popek and Geoffrey Kuenning, *Replication Requirements in Mobile Environments*, Mobile Networks and Applications, Vol. 6, No. 6, pp. 525-533, 2001.
7. G. Ray, J. R. Harista, and S. Seshadri, *Database compression: A performance enhancement tool*, in Proceedings of the 7th International Conference on Management of Data (COMAD), Pune, India, 1995.
8. T. Westmann, D. Kossman, S. Helmer, G. Moerkotte, *The implementation and performance of compressed databases*, Sigmod Record, 29 (3), Sept., 2000, pp 55-67.
9. G. Klir, *Reconstructability Analysis: An Offspring of Ashby's Constraint Theory*, Systems Research, vol. 3 (4), pp. 267-271, 1986.
10. D. Paul Benjamin and Adrian Walker, *Semantic Encoding and Compression of Database Tables*, U.S. Patent 6,691,132.