Master’s Project Proposal

Ensuring data integrity with tamper evident encryption of integers using keyed Hash Message Authentication Code

Brad Baker
September 28, 2009

Approved by:

Advisor: Dr. Edward Chow                     Date

Committee Member: Dr. Jugal Kalita               Date

Committee Member: Dr. Terrance Boult           Date
1. Abstract

This project will design and implement improvements to a Hash Message Authentication Code (HMAC) based encryption scheme presented in [1] “How to Construct a New Encryption Scheme Supporting Range Queries on Encrypted Database” presented by Dong Hyeok Lee, You Jin Song, Sung Min Lee, Taek Yong Nam and Jong Su Jang at the IEEE 2007 International Conference on Convergence Information Technology. The result will be a symmetric encryption scheme based on the HMAC operation using the SHA-1 hash algorithm which will have the ability to detect unauthorized updates to stored ciphertext data. In addition to tamper detection, efficiency improvements will be included in the encryption algorithm.

This improved encryption scheme will be designed to process positive integer values and will decompose these values into buckets using modular arithmetic. The buckets will be encrypted using the HMAC-SHA1 function. The secret key will be expanded for each plaintext value using a specific transformation process. Data will be processed through block encryption and decryption, with the goal of implementing the scheme in a database system. Unauthorized changes to ciphertext values will be detected. The hash output of the HMAC function will serve as both the ciphertext and digest. Figure 1 shows the concept of the encryption algorithm.

![HMAC based Tamper Evident Encryption (HTEE) Concept](image)

**Figure 1 - Concept of the proposed project algorithm**

- **$U$**: Unique Value: A value such as DB primary key
- **$P$**: Plaintext: An unencrypted value
- **$B$**: Bucket: A partial plaintext value
- **$K$**: Key: A secret key value
- **$C$**: Ciphertext: An encrypted value
- **$f(K)$ and $f(P)$**: Functions that modify Key or Plaintext values
- **HMAC($B, K$)**: A Hashed Message Authentication Code function using Bucket and Key values
This paper is organized as follows: the background and motivation for the project are discussed in section two, the concepts for algorithm improvements are discussed in section three, and specific project details are discussed in section four.

2. Introduction and background

This project relies on existing work including the Hash Message Authentication Code (HMAC) ([3], [8]) and a HMAC based encryption algorithm [1]. This section summarizes concepts from existing work that motivates this project.

2.1 General introduction

Confidentiality and integrity of data are important and continuing goals in computing, particularly in database processing and communications. The amount and importance of sensitive data that computers store, process and transmit over networks is constantly increasing, as are the consequences of data breaches. Confidentiality ensures that sensitive data is not revealed to unauthorized individuals, and integrity ensures that sensitive data is not corrupted or updated by unauthorized individuals. Cryptography provides several processes to support confidentiality and integrity, including symmetric and asymmetric key algorithms for confidentiality and digest or signature algorithms for integrity. Generally, encryption algorithms such as AES or RSA provide strong confidentiality but don’t provide integrity and hash digest algorithms such as SHA or Whirlpool provide integrity without confidentiality. Traditional methods to obtain both confidentiality and integrity include combining encryption and digest algorithms, for example encrypting the plaintext data and computing a digest against the ciphertext.

Considering hash digest algorithms, one challenge is that an attacker can recalculate and update the hash digest after changing the data. For example, if a database has a hash digest appended to each record, the attacker can update the hash after changing stored data, and data integrity is lost. This highlights an underlying requirement of hash algorithms that the digest must be protected in a secure channel. Once the digest is computed, it must be stored in a trusted location where it can’t be updated. Because it is infeasible to store database record hash digests in a trusted external location, one solution for a database environment is to encrypt the data record and the digest together, preventing an attacker from recalculating the digest after an update. Another solution is the use of a message authentication code, such as the keyed Hash Message Authentication Code (HMAC). Message authentication codes such as HMAC provide the function of a digest that is protected from unauthorized updates with a secret key.

2.2 Summary of HMAC

Keyed HMAC, or Hash Message Authentication Code ([3], [8]), is a process that uses a secret key and a hash algorithm such as MD5 or SHA to generate a message authentication code. This process is symmetric, so two parties communicating with HMAC must share the same secret key. By using a hash algorithm in conjunction with a key, it prevents an unauthorized user from modifying the message or the digest without being detected. This can protect against man-in-the-middle attacks on the message, but it is not designed to encrypt the message itself; only protect it from unauthorized update. HMAC can be defined as a function that takes a key and a plaintext message as input. Any hash algorithm can be used, including MD5, SHA-1, SHA-256, etc. The HMAC algorithm defines two padding constants, the inner pad and the outer pad, with values (0x3636…) and (0x5c5c…) respectively, each expanded to the block size of the hash algorithm. To calculate the HMAC, first the exclusive-or of the key and the input pad is found. This result is appended to the beginning of the message to be processed. The result is then hashed with the chosen hash algorithm, producing an intermediate digest. In the next step, the
exclusive-or of the key and the output pad is found, and that result is appended to the beginning of the intermediate digest. The result is hashed again, producing the final message authentication code. This operation is summarized in Figure 1, where \( \oplus \) denotes exclusive-or, \( ++ \) denotes concatenation, \( \{K\} \) is the secret key, \( \{m\} \) is the plaintext message, and \( \{H\} \) is the hash function.

\[
\text{HMAC}(K, m) = \mathcal{H}\{(K \oplus \text{ipad}) ++ \mathcal{H}\{(K \oplus \text{ipad}) ++ m\}\}.
\]

Figure 2 – HMAC operation

Each calculation of the HMAC digest requires running the underlying hash function twice. The output of HMAC is a binary code, equal in length to the hash function digest. This code can only be reproduced with the same key and message, and the cryptographic strength is based on the strength of the hash algorithm, which can be modified if required.

2.3 Previous work

This project builds on the encryption scheme proposed in [1] “How to Construct a New Encryption Scheme Supporting Range Queries on Encrypted Database” presented by Dong Hyeok Lee, You Jin Song, Sung Min Lee, Taek Yong Nam and Jong Su Jang at the IEEE 2007 International Conference on Convergence Information Technology. This encryption scheme processes integer values in a database environment, using the keyed Hash Message Authentication Code (keyed HMAC) for encryption. Plaintext values are decomposed into a numeric bucket and remainder value using modular integer arithmetic and each value is encrypted and stored separately. The encryption process involves calculating the HMAC digest using a secret key with the bucket and remainder values. The ciphertext result is two hash digest values, which are stored in the database. The decryption transformation involves an exhaustive search across all possible bucket and remainder values using HMAC and the secret key, where a match indicates that the plaintext has been found.

A detailed analysis of this scheme can be found in my July, 2009 paper [2] “Analysis of an HMAC Based Database Encryption Scheme”. The proposed algorithm is unique in the way that it applies the HMAC calculation to encrypt integer data, rather than simply storing the digest of encrypted data. In this way, the digest becomes the encrypted ciphertext. Normally hash digest calculations are one way functions and to not have inverses, which presents a problem when trying to decrypt hash ciphertext data. The proposed algorithm solves this problem by searching through all possible bucket and remainder values using the secret key and HMAC calculation to find matching ciphertext.

In the analysis of this algorithm, two problems were found. The first problem is inefficiency, when using a single bucket and single remainder for large integer values it is possible that HMAC will need to be calculated up to a maximum of 2,000,000 times for a single decryption operation of a integer value in the range of \( 1 \times 10^{12} \). The second problem is that hash output from HMAC does not preserve any information from the plaintext, and range queries or other non-equality queries over encrypted data do not work. The use of range queries over encrypted data was a goal of the encryption scheme.

3. Concept for algorithm improvement

This project will implement improvements to the algorithm presented in [1], with the addition of tamper detection replacing the concept of range queries over encrypted data. This project will develop an improved HMAC based integer encryption algorithm, with primary goals being improved efficiency, data integrity / tamper detection, and secure encryption. The following sections summarize the planned improvements.
3.1 Improvement to bucket processing

The improved algorithm will generalize the concept of bucket and residual decomposition of the plaintext value. Rather than creating one bucket and one residual, each of which have a wide range of possible values, a number of buckets equal to \( p = \text{floor}(\log_{1000}(\text{Plaintext})) + 1 \) will be used. This value will be referred to as \( \{p\} \). By using the floor function with the log base 1000, we can find the scale of the plaintext with a bucket size of 1000. For example, the integer value 122,344,566,788,900 will be decomposed into five buckets with values 122, 344, etc. These buckets are found using a modulus calculation algorithm as shown in the following pseudocode.

```plaintext
residual = plaintext
P = \text{floor}(\log_{1000}(\text{plaintext})) + 1
while (P >= 0)
{
    bucket = (residual - residual mod 1000^P) / 1000^P
    residual = residual mod 1000^P
    P = P - 1
}
```

![Figure 3 – Bucket decomposition for bucket size 1000](image_url)

The plaintext value for each power of 1000 is found, relating to the trillions, billions, millions, thousands and ones locations in the original integer. Each of these bucket values will be encrypted with HMAC separately. This increased number of buckets in the decomposition step will provide the first improvement to the encryption algorithm. This is because the search process in the decryption transformation will only need to search through a maximum of 1000 values over 5 buckets, or 5000 HMAC operations in the decryption search per plaintext value. In the original scheme, two buckets of size 1,000,000 might be used, requiring a maximum search of 2,000,000 HMAC operations per plaintext value. By using more buckets, each with a smaller range of values makes the decryption process more efficient in processing. The tradeoff is storage of more ciphertext data since five hashed bucket values are used instead of two.

3.2 Key transformation process

The original proposed encryption scheme used a single shared key for all HMAC encryption operations. By using a single key for encryption of all records it is difficult or impossible to detect tampering with ciphertext data. Consider a data record that contains indexing information such as customer ID in addition to encrypted numeric data. If all records are encrypted individually using the same secret key, an attacker can swap ciphertext values from different rows, an update that can’t be detected using the ciphertext alone. The only way that an administrator can detect the unauthorized update is through external information or database audit logs. In addition to the data integrity problem, a single secret key will reveal equal plaintext values because their ciphertext values will be identical.

For this project, a method of key transformation will be developed that will both obscure identical plaintext values and tie ciphertext values to the order of encryption, so any change in stored ciphertext can be detected. Two key transformations will be defined; the first will change the original secret key for each plaintext value and the second will change the secret key for each bucket of a given plaintext value. These transformations will result in a different key being used for each bucket of each plaintext in the encryption process. The first transformation, called the element key, will be based on the order of processing so the key sequence from 1 through N element keys can be reproduced without valid ciphertext of plaintext values. The second
transformation, called the bucket key, will be based on the ciphertext output of other bucket values, so it cannot be reproduced without valid ciphertext data.

By using different key values for each encryption operation and a small range of valid plaintext values for each ciphertext, any update to the ciphertext data will cause the decryption process to fail. A lack of result for decryption indicates that the ciphertext has been tampered with, which will lead the administrator to research audit logs and reconstruct the original data. Although this method will detect tampering with the ciphertext, it will not be able to restore the original plaintext data. The important feature of this encryption method that supports tamper detection is the ratio between the number of plaintext values (1000), the number of possible keys ($2^{512}$) and the number of possible HMAC output values ($2^{160}$). These numbers are based on the block size and output size of the SHA-1 hash algorithm when used for HMAC. If any ciphertext is changed, it is essentially impossible that the transformed key matching that ciphertext will decrypt to a different plaintext value. The tamper detection process is reliant on the block encryption and decryption of data values. If any plaintext value needs to change, all records must be decrypted and re-encrypted.

The element key transformation takes the prior element key and the initial secret key as input values. This transformation calculates the HMAC digest using a portion of the prior element key as input message and the initial secret key as the HMAC key. This hashed output is appended to the beginning of the prior element key and the result is truncated to fit the HMAC key length, 512 bits when using the SHA-1 algorithm. The bucket key transformation is similar to the element key transformation except that the prior bucket ciphertext value is concatenated to the element key instead of an intermediate HMAC output. Figure #3 shows the concept of the improved encryption transformation, including the multiple bucket decomposition and the key transformation process. In the below figure, {P} represents plaintext, {B} represents decomposed bucket values, {C} represents ciphertext values and {K} represents secret key values.
4. Project plan

The ultimate goal of this project is to define improvements to the HMAC encryption algorithm, implement and test those improvements. The end result will be a working tamper detection encryption algorithm. Additional goals include building the process into an add-on for a DBMS such as Postgresql, so the process can be easily used in a database environment.

4.1 Schedule

This project is to be completed by the end of term, fall 2009.

The following is a proposed schedule for the project:

August 28, 2009       Completed final project proposal
4.2 Deliverables

There are several deliverables as goals for this project. These are presented in the order of highest to lowest priority. In the situation of technical problems, lower priority deliverables may not be realized.

- High priority – critical:
  - Project report, including:
    - A complete discussion of the project and results
  - Improved encryption/decryption algorithm, including:
    - The ability to detect tampering with ciphertext.
    - Efficiency improvements.
    - Central concepts of bucket based processing and key transformation.
  - Implementation of the improved algorithm for flat file processing, including:
    - A program to encrypt and decrypt plaintext and ciphertext values stored in normal text files
  - Analysis of improved algorithm, including:
    - Quantification of the strength of tamper protection
    - Quantification of the strength of encryption
    - Possibly comparison to AES encryption with block chaining mode or similar
  - Testing of improved algorithm and implementation, including:
    - Efficiency testing and tamper detection testing.
    - Thorough testing to validate the results.
  - Implementation of the improved algorithm as a DBMS add-on, including:
    - A program to encrypt and decrypt plaintext and ciphertext values stored in database tables.

- Medium priority – goals:
  - Improved algorithm to handle additional challenges, pending feasibility
    - This can include processing negative numbers, some real numbers, or other improvements that might become apparent

4.3 References

Preliminary references relating to research done for this project proposal are listed below.

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4420452&isnumber=4420217

URL: http://cs.uccs.edu/~gsc/pub/master/bbaker/doc/final_paper_bbaker_cs592.doc


URL: http://doi.acm.org/10.1145/1103780.1103784

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1649595&isnumber=34591

URL: http://doi.acm.org/10.1145/948109.948156