Ensuring data integrity with tamper evident encryption of integers using keyed Hash Message Authentication Code
Confidentiality and integrity of data are important features needed in a database environment.

Standard solutions exist including:

- Symmetric key encryption like AES and hash digests like SHA-1

Standard solutions require end-user to build a custom process combining hash and encryption functions.

This project presents the “HMAC based Tamper Evident Encryption” scheme (HTEE) as an alternative solution.
The motivation for this project is:

- To provide a one-step encryption and tamper detection process for the end user
- To provide a one-column solution to confidentiality and integrity
- To provide a degree of performance improvement
- To improve on a previously published work which introduced the HMAC based encryption/decryption process.
- To explore alternative uses to hash digests and HMAC including the ability to invert the digest
As a summary, this project:

- Will improve an encryption scheme proposed in an existing work [1]
- Will add efficiency and tamper detection features
- Will encrypt integers using a hash-based process
- Will decrypt ciphertext using an exhaustive search process
- Will implement the improved algorithm
- Will test and analyze the improved algorithm
Prior work, Summer 2009:

- Studied, implemented and tested a HMAC based encryption scheme proposed in [1]
- The scheme operated on integer plaintext values, breaking them into a bucket and remainder
- The scheme did not support range queries, and was inefficient in the design for encryption and decryption.
- A detailed analysis and results are available in [2]
The original HMAC based encryption scheme appears to be unique: it uses a hash for two-way encryption

- This is done by limiting the plaintext domain and performing exhaustive decryption searches
- The strength of the algorithm is variable per implemented hash function
- HMAC is a keyed-hash algorithm, see [3] and [4]

This project will improve the proposed algorithm

- Efficiency updates will be included
- The use of hash ciphertext will be leveraged to provide tamper detection of ciphertext values
Summary of efficiency improvements:

- The original algorithm decomposes integers into a single bucket and remainder with modulus.
- The improved algorithm will decompose integers into buckets of size 1000:
  - The ones, thousands, millions, billions, trillions, etc. values will each be buckets.
- For example, a plaintext value of one trillion will produce five HMAC outputs as ciphertext.
- The smaller plaintext range is much more efficient in decryption, but it produces extra ciphertext output.
- The encryption function is redefined to decrease the HMAC operations.
Summary of tamper detection

- Ciphertext values provide basic tamper detection preventing random tampering
  - Will use a bucket size of 1000 and the SHA1 hash algorithm
  - 1000 plaintext values will be combined with $2^{512}$ key values, resulting in $2^{160}$ ciphertext values
  - With the ratio between plaintext and ciphertext it is improbable that a change in ciphertext will result in a different plaintext
- A key transformation process will be included to prevent interchange of ciphertext values
  - A deterministic processing-order based or unique-value based transformation process will be defined
This image shows the concept of the improved algorithm

Note the multiple buckets

Note the key transformation between plaintext and bucket values

The decryption process searches among plaintext values for a match to ciphertext

---

**Tamper Evident HMAC Encryption Concept**

- **P**: Plaintext: A unencrypted value
- **B**: Bucket: A partial plaintext value
- **K**: Key: A secret key value
- **C**: Ciphertext: A 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

---

Brad Baker - Master's project proposal  9/28/2009
The goals for this project are:

• Finalize the improved algorithm:
  • Including bucket processing and key transformation
• Implement the encryption scheme:
  • Using a command line utility for flat file processing
  • As a database add-on for the Postgresql DBMS
• Test the implementation:
  • Ensure validity, performance and tamper detection
• Analyze the improved algorithm:
  • Quantify cryptographic and tamper detection strength
• Project report:
  • Produce a comprehensive report of the project
This project will 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
- September 1, 2009: Begin project work
- October 1, 2009: Approved project proposal
- November 1, 2009: Completed project and draft report
- November 6, 2009: Completed final project report
- Before Nov 24, 2009: Completed project defense
- December 19, 2009: End of fall 2009 term


