Encryption

Table of Contents

About Encryption

Encryption is a means of modifying plain-text into an unreadable format. Algorithms used to encrypt text are known as ciphers. For example, replacing each letter in a message with another letter in the alphabet is a simple form of encryption known as a Caesar Cipher. Today's algorithms are much more advanced, though many systems still rely on proprietary ciphers that are no more advanced than a simple Caesar Cipher. Blesta, however, only uses properly vetted, and verifiably secure, open encryption ciphers.

Why Use Encryption?

Assuming your database server is properly secured, access to the database is strictly enforced, and all software running on or connecting to the database is completely bug free, encryption is unnecessary. Unfortunately, reality dictates that this is impossible to achieve. So to ensure that in the event unauthorized access to the data is obtained, no matter how unlikely, it becomes necessary to encrypt that data to ensure it remains protected.

Why Not Encrypt Everything?

There are two main draw-backs to encryption:

1. Encrypting/Decrypting data consumes CPU time.
2. Encrypted data can not be searched or indexed.

For the reasons above, it typically only makes sense to encrypt sensitive data.

Encryption Methods

Block Cipher

A block cipher is an encryption method that uses the same key for both encryption and decryption.

AES-256

Blesta uses AES-256 for all block cipher requirements. The key for this encryption method is derived from an HMAC SHA-256 hash of the Blesta.system_key configuration setting.

Examples

- Custom client fields values
- Credit card expiration dates
- Credit card last-four digits
- Bank account last-four digits
- Module field values
- Service field values
- Gateway field values
- Company settings
- System settings

Asymmetric Cipher

An asymmetric cipher, also known as a public-key cipher, is an encryption method that uses one key for encryption and another for decryption.

RSA

Blesta uses RSA with a key length of up to 3072-bits for all asymmetric cipher requirements. The keys for this encryption method are generated for each company in the system, at the time the company is created.

The public key is stored in plain-text and is used only to encrypt data. Data encrypted with this key may only be decrypted using the private key. The private key is stored in the database encrypted using AES-256. If a passphrase is chosen, the passphrase is used as the key to the HMAC SHA-256 hash which is then used as the AES key to decrypt the private key. Because the passphrase is not stored anywhere, this makes it impossible to decrypt data encrypted with the public key without providing the passphase. Because of the nature of public key ciphers, data may be encrypted without knowing the private key.

Examples

- Credit card numbers
- Bank account numbers
- Bank routing numbers

Key-Based Hashing

A key-based hashing algorithm, also known as hash-based message authentication is a means of creating a message digest of a fixed-length from an arbitrary-length plain-text using a one-way algorithm. That is, once the message digest is computed the plain-text is impossible to decipher. Hashes are not a form of encryption, but are often used in places where decrypting the plain-text is unnecessary. For example, in validating a given password is correct.

HMAC SHA-256

Blesta uses HMAC SHA-256 for many different hashing requirements including:

- Generating AES-256 encryption keys
- Hashing passwords before being bcrypted

bcrypt HMAC SHA-256

Blesta uses bcrypt to store passwords for authentication. The bcrypt algorithm is a slow-computing algorithm that is designed to take a small, but significant amount of time to generate a result. This makes it computationally inefficient to brute-force.The amount of work involved in computing the bcrypt result is controlled by the Blesta.hash_work configuration setting.

Before a password is hashed using bcrypt, however, it is hashed using HMAC SHA-256. The HMAC SHA-256 process produces a 256-bit (64-hexadecimal character) string, which is then hashed using bcrypt. This extra step provides additional security for short passwords, extremely long passwords (see denial of service), and dictionary attacks.