Key Derivation Function Explanations

Key Derivation Functions (KDF) are mechanisms by which human-readable information, usually a password or other secret information, is translated into a cryptographic key suitable for data protection. For further information, read the Wikipedia entry on Key Derivation Functions.

Currently, KDFs are ingested by CipherProvider implementations and return a fully-initialized Cipher object to be used for encryption or decryption. Due to the use of a CipherProviderFactory, the KDFs are not customizable at this time. Future enhancements will include the ability to provide custom cost parameters to the KDF at initialization time. As a work-around, CipherProvider instances can be initialized with custom cost parameters in the constructor but this is not currently supported by the CipherProviderFactory.

Here are the KDFs currently supported by NiFi (primarily in the EncryptContent processor for password-based encryption (PBE)) and relevant notes:

- **NiFi Legacy KDF**
  - The original KDF used by NiFi for internal key derivation for PBE, this is 1000 iterations of the MD5 digest over the concatenation of the password and 16 bytes of random salt.
  - This KDF is **deprecated as of NiFi 0.5.0** and should only be used for backwards compatibility to decrypt data that was previously encrypted by a legacy version of NiFi.

- **OpenSSL PKCS#5 v1.5 EVP_BytesToKey**
  - This KDF was added in v0.4.0.
  - This KDF is provided for compatibility with data encrypted using OpenSSL’s default PBE, known as EVP_BytesToKey. This is a single iteration of MD5 over the concatenation of the password and 8 bytes of random ASCII salt. OpenSSL recommends using PBKDF2 for key derivation but does not expose the library method necessary to the command-line tool, so this KDF is still the de facto default for command-line encryption.

- **Bcrypt**
  - This KDF was added in v0.5.0.
  - Bcrypt is an adaptive function based on the Blowfish cipher. This KDF is strongly recommended as it automatically incorporates a random 16 byte salt, configurable cost parameter (or "work factor"), and is hardened against brute-force attacks using GPGPU (which share memory between cores) by requiring access to "large" blocks of memory during the key derivation. It is less resistant to FPGA brute-force attacks where the gate arrays have access to individual embedded RAM blocks.
  - Because the length of a Bcrypt-derived key is always 184 bits, the complete output is then fed to SHA-512 digest and truncated to the desired key length. This provides the benefit of the avalanche effect on the formatted input.
  - The recommended minimum work factor is 12 ($2^{12}$ key derivation rounds) (as of 2/1/2016 on commodity hardware) and should be increased to the threshold at which legitimate systems will encounter detrimental delays (see schedule below or use BcryptCipherProviderGroovyTest#testDefaultConstructorShouldProvideStrongWorkFactor() to calculate safe minimums).
  - The salt format is $s0$.$e010$ABCDEFGHIJKLMNOPQRSTUV. The salt is delimited by 's' and the three sections are as follows:
    - $s0$ - the version of the format. An extensive explanation can be found here. NiFi currently uses 's0' for all salts generated internally.
    - $e0$ - the work factor. This is actually the $log_2$ value, so the total iteration count would be $2^{10}$ in this case.
    - ABDCEFHIJKLMNOPQRSTUVWXYZ - the 22 character, Base64-encoded, unpadded, raw salt value. This decodes to a 16 byte salt used in the key derivation.

- **Scrypt**
  - This KDF was added in v0.5.0.
  - Scrypt is an adaptive function designed in response to bcrypt. This KDF is recommended as it requires relatively large amounts of memory for each derivation, making it resistant to hardware brute-force attacks.
  - The recommended minimum cost is $N=2^{14}$, $r=8$, $p=1$ (as of 2/1/2016 on commodity hardware) and should be increased to the threshold at which legitimate systems will encounter detrimental delays (see schedule below or use ScryptCipherProviderGroovyTest#testDefaultConstructorShouldProvideStrongParameters() to calculate safe minimums).
  - The salt format is $s2a$.$s10$ABCDCEFHIJKLMNOPQRSTUVWXYZ. The salt is delimited by 's' and the three sections are as follows:
    - $s2a$ - the version of the format. NiFi currently uses 's2a' for all salts generated internally.
    - $s10$ - the cost parameters. This is actually a hexadecimal encoding of $N$, $r$, $p$ using shifts. This can be formed/parsed using ScryptEncodeParams() and ScryptParseParameters().
    - ABDCEFHIJKLMNOPQRSTUVWXYZ - the 11-44 character, Base64-encoded, unpadded, raw salt value. This decodes to a 8-32 byte salt used in the key derivation.

- **PBKDF2**
  - This KDF was added in v0.5.0.
  - Password-Based Key Derivation Function 2 is an adaptive derivation function which uses an internal pseudorandom function (PRF) and iterates it many times over a password and salt (at least 16 bytes).
  - The PRF is recommended to be HMAC-SHA-256 or HMAC-SHA-512. The use of an HMAC cryptographic hash function mitigates a length extension attack.
  - The recommended minimum number of iterations is 160,000 (as of 2/1/2016 on commodity hardware). This number should be doubled every two years (see schedule below or use PBKDF2CipherProviderGroovyTest#testDefaultConstructorShouldProvideStrongIterationCount() to calculate safe minimums).
  - This KDF is not memory-hard (can be parallelized massively with commodity hardware) but is still recommended as sufficient by NIST SP 800-132 and many cryptographers (when used with a proper iteration count and HMAC cryptographic hash function).
• None
  • This KDF was added in v0.5.0.
  • This KDF performs no operation on the input and is a marker to indicate the raw key is provided to the cipher. The key must be provided in hexadecimal encoding and be of a valid length for the associated cipher/algorithm.

Resources

• Explanation of optimal scrypt cost parameters and relationships
• NIST Special Publication 800-132
• OWASP Password Storage Work Factor Calculations
• PBKDF2 rounds calculations
• Scrypt as KDF vs password storage vulnerabilities
• Scrypt vs. Bcrypt (as of 2010)
• Bcrypt vs PBKDF2
• Choosing a work factor for Bcrypt
• Spring Security Bcrypt
• OpenSSL EVP BytesToKey PKCS#1v1.5
• OpenSSL PBKDF2 KDF
• OpenSSL KDF flaws description