Key Derivation for Authentication, Integrity, and Privacy

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working
documents of the Internet Engineering Task Force (IETF), its areas,
and its working groups. Note that other groups may also distribute
working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as ‘‘work in progress.’’

To learn the current status of any Internet-Draft, please check the
‘‘1id-abstracts.txt’’ listing contained in the Internet-Drafts Shadow
Directories on ds.internic.net (US East Coast), nic.nordu.net
(Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific
Rim).

Distribution of this memo is unlimited. Please send comments to the
author.

Abstract

Recent advances in cryptography have made it desirable to use longer
cryptographic keys, and to make more careful use of these keys. In
particular, it is considered unwise by some cryptographers to use the
same key for multiple purposes. Since most cryptographic-based
systems perform a range of functions, such as authentication, key
exchange, integrity, and encryption, it is desirable to use different
cryptographic keys for these purposes.

This document does not define a particular protocol, but defines
a set of cryptographic transformations for use with arbitrary network
protocols and block cryptographic algorithm.

Deriving Keys

In order to use multiple keys for different functions, there are two
possibilities:

- Each protocol ‘‘key’’ contains multiple cryptographic keys. The
  implementation would know how to break up the protocol ‘‘key’’ for
  use by the underlying cryptographic routines.

- The protocol ‘‘key’’ is used to derive the cryptographic keys.
  The implementation would perform this derivation before calling
the underlying cryptographic routines.

In the first solution, the system has the opportunity to provide separate keys for different functions. This has the advantage that if one of these keys is broken, the others remain secret. However, this comes at the cost of larger "keys" at the protocol layer. In addition, since these "keys" may be encrypted, compromising the cryptographic key which is used to encrypt them compromises all the component keys. Also, the not all "keys" are used for all possible functions. Some "keys", especially those derived from passwords, are generated from limited amounts of entropy. Wasting some of this entropy on cryptographic keys which are never used is unwise.

The second solution uses keys derived from a base key to perform cryptographic operations. By carefully specifying how this key is used, all of the advantages of the first solution can be kept, while eliminating some disadvantages. In particular, the base key must be used only for generating the derived keys, and this derivation must be non-invertible and entropy-preserving. Given these restrictions, compromise of one derived keys does not compromise the other subkeys. Attack of the base key is limited, since it is only used for derivation, and is not exposed to any user data.

Since the derived key has as much entropy as the base keys (if the cryptosystem is good), password-derived keys have the full benefit of all the entropy in the password.

To generate a derived key from a base key:

\[ \text{Derived Key} = \text{DK(Base Key, Well-Known Constant)} \]

where

\[ \text{DK(Key, Constant)} = \text{n-fold(E(Key, Constant))} \]

In this construction, E(Key, Plaintext) is a block cipher, Constant is a well-known constant defined by the protocol, and n-fold is an algorithm which takes m input bits and "stretches" them to form n output bits with no loss of entropy, as described in [Blumenthal96]. In this document, n-fold is always used to produce n bits of output, where n is the key size of E.

If the output of E is is shorter than n bits, then some entropy in the key will be lost. If Constant is not a multiple of the block size of E, then Constant must be padded so it may be encrypted. If the Constant is larger than the block size, then it must be folded down to the block size to avoid chaining, which affects the distribution of entropy.

In any of these situations, a variation of the above construction is used, where the Constant is encrypted in Counter Output Mode:

\[ \text{DK(Key, Constant)} = \text{n-fold(E-COM(Key, Constant))} \]
Constant is padded to the next multiple of the block size of \( E \) with zero octets. If the Constant is a multiple of the block size, but less than the key size, then a full block of octets must be appended. If the resulting padded Constant is larger than a single block, the n-fold algorithm must be used to fold the constant down to the number of bits in a single block. Next, \( E(\text{Key}, \text{padded Constant}) \) is computed. If the output of this encryption is less than the key size, then the padding octets are each incremented (yielding some number of octets having the value one), and \( E(\text{Key}, \text{padded Constant}) \) is computed again and appended to the result. This process is repeated until the output result is larger than the key size. (Note that with some cryptosystems, such as DES ECB, this is equivalent to concatenating the padded Constants and encrypting once.)

This output is then n-folded to produce the derived key.

Since the derived key is the result of an encryption the base key, deriving the base key from the derived key is equivalent to determining the key from a single plaintext/ciphertext pair. Thus, this construction is as strong as the cryptosystem itself, as long as no entropy is lost in this transformation.

Deriving Keys from Passwords

When protecting information with a password or other user data, it is necessary to convert an arbitrary bit string into an encryption key. In addition, it is sometimes desirable that the transformation from password to key be difficult to reverse. A simple variation on the construction in the prior section can be used:

\[
\text{Key} = \text{DK}(\text{n-fold(Password)}, \text{Well-Known Constant})
\]

The n-fold algorithm is reversible, so recovery of the n-fold output is equivalent to recovery of Password. However, recovering the n-fold output is difficult for the same reason recovering the base key from a derived key is difficult.

Traditionally, the transformation from plaintext to ciphertext, or vice versa, is determined by the cryptographic algorithm and the key. A simple way to think of derived keys is that the transformation is determined by the cryptographic algorithm, the constant, and the key. Another way to think of derived keys is to think of them as derived for interoperability, the constants used to derive keys for different purposes must be specified in the protocol specification. The constants must not be specified on the wire, or else an attacker who determined one derived key could spoof data using that derived key, rather than the one the protocol designer intended.

Determining which parts of a protocol require their own constants is an issue for the designer of protocol using derived keys.
Security Considerations

This entire document deals with security considerations relating to the use of cryptography in network protocols.

Acknowledgements

I would like to thank Uri Blumenthal and Bill Sommerfeld for their contributions to this document.

References


Author’s Address

Marc Horowitz
Cygnus Solutions
955 Massachusetts Avenue
Cambridge, MA 02139

Phone: +1 617 354 7688
Email: marc@cygnus.com