Internet Research Task Force (IRTF)
Request for Comments: 8645
Category: Informational
ISSN: 2070-1721

S. Smyshlyaev, Ed.
CryptoPro

August 2019

Re-keying Mechanisms for Symmetric Keys

Abstract

A certain maximum amount of data can be safely encrypted when encryption is performed under a single key. This amount is called the "key lifetime". This specification describes a variety of methods for increasing the lifetime of symmetric keys. It provides two types of re-keying mechanisms based on hash functions and block ciphers that can be used with modes of operations such as CTR, GCM, CBC, CFB, and OMAC.

This document is a product of the Crypto Forum Research Group (CFRG) in the IRTF.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Crypto Forum Research Group of the Internet Research Task Force (IRTF). Documents approved for publication by the IRSG are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8645.

Table of Contents

1. Introduction

A certain maximum amount of data can be safely encrypted when encryption is performed under a single key. Hereinafter, this amount will be referred to as the "key lifetime". The need for such a limitation is dictated by the following methods of cryptanalysis:

1. Methods based on the combinatorial properties of the used block cipher mode of operation

   These methods do not depend on the underlying block cipher. Common mode restrictions derived from such methods are of order 2^{n/2}, where n is a block size defined in Section 3. [Sweet32] includes an example of an attack that is based on such methods.

2. Methods based on side-channel analysis issues

   In most cases, these methods do not depend on the used encryption modes and weakly depend on the used cipher features. Limitations resulting from these considerations are usually the most restrictive ones. [TEMPEST] is an example of an attack that is based on such methods.

3. Methods based on the properties of the used block cipher

   The most common methods of this type are linear and differential cryptanalysis [LDC]. In most cases, these methods do not depend on the used modes of operation. In the case of secure block ciphers, bounds resulting from such methods are roughly the same as the natural bounds of 2^n and are dominated by the other bounds above. Therefore, they can be excluded from the considerations here.

As a result, it is important to replace a key when the total size of the processed plaintext under that key approaches the lifetime limitation. A specific value of the key lifetime should be determined in accordance with some safety margin for protocol security and the methods outlined above.

Suppose L is a key lifetime limitation in some protocol P. For simplicity, assume that all messages have the same length m. Hence, the number of messages q that can be processed with a single key K should be such that m * q <= L. This can be depicted graphically as a rectangle with sides m and q enclosed by area L (see Figure 1).

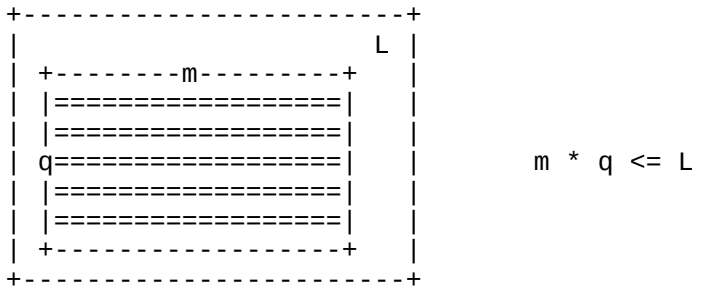


Figure 1: Graphic Display of the Key Lifetime Limitation

In practice, the amount of data that corresponds to limitation L may not be enough. The simplest and obvious solution in this situation is a regular renegotiation of an initial key after processing this threshold amount of data L. However, this reduces the total performance, since it usually entails termination of application data transmission, additional service messages, the use of a random number generator, and many other additional calculations, including resource-intensive public key cryptography.

For protocols based on block ciphers or stream ciphers, a more efficient way to increase the key lifetime is to use various re-keying mechanisms. This specification considers re-keying mechanisms for block ciphers only; re-keying mechanisms typical for stream ciphers (e.g., [Pietrzak2009], [FPS2012]) are beyond the scope of this document.

Re-keying mechanisms can be applied at the different protocol levels: the block cipher level (this approach is known as fresh re-keying and is described, for instance, in [FRESHREKEYING]; the block cipher mode of operation level (see Section 6); and the protocol level above the block cipher mode of operation (see Section 5). The usage of the first approach is highly inefficient due to the key changing after each message block is processed. Moreover, fresh re-keying mechanisms can change the block cipher internal structure and, consequently, can require an additional security analysis for each particular block cipher. As a result, this approach depends on particular primitive properties and cannot be applied to any arbitrary block cipher without additional security analysis. Therefore, fresh re-keying mechanisms go beyond the scope of this document.

Thus, this document contains the list of recommended re-keying mechanisms that can be used in the symmetric encryption schemes based on the block ciphers. These mechanisms are independent from the

particular block cipher specification, and their security properties rely only on the standard block cipher security assumption.

This specification presents two basic approaches to extending the lifetime of a key while avoiding renegotiation, which were introduced in [AAOS2017]:

1. External re-keying

   External re-keying is performed by a protocol, and it is independent of the underlying block cipher and the mode of operation. External re-keying can use parallel and serial constructions. In the parallel case, data processing keys K^1, K^2, ... are generated directly from the initial key K independently of each other. In the serial case, every data-processing key depends on the state that is updated after the generation of each new data-processing key.

   As a generalization of external parallel re-keying, an external tree-based mechanism can be considered. It is specified in Section 5.2.3 and can be viewed as the tree generalization in [GGM]. Similar constructions are used in the one-way tree mechanism ([OWT]) and [AESDUKPT] standard.

2. Internal re-keying

   Internal re-keying is built into the mode, and it depends heavily on the properties of the mode of operation and the block size.

The re-keying approaches extend the key lifetime for a single initial key by allowing the leakages to be limited (via side channels) and by improving the combinatorial properties of the used block cipher mode of operation.

In practical applications, re-keying can be useful for protocols that need to operate in hostile environments or under restricted resource conditions (e.g., those that require lightweight cryptography, where ciphers have a small block size that imposes strict combinatorial limitations). Moreover, mechanisms that use external or internal re-keying may provide some protection against possible future attacks (by limiting the number of plaintext-ciphertext pairs that an adversary can collect) and some properties of forward or backward security (meaning that past or future data-processing keys remain secure even if the current key is compromised; see [AbBell] for more details). External or internal re-keying can be used in network protocols as well as in the systems for data-at-rest encryption.

Depending on the concrete protocol characteristics, there might be situations in which both external and internal re-keying mechanisms (see Section 7) can be applied. For example, a similar approach was used in Taha's tree construction (see [TAHA]).

Note that there are key-updating (key regression) algorithms (e.g., [FKK2005] and [KMNT2003]) that are called "re-keying" as well, but they pursue goals other than increasing the key lifetime. Therefore, key regression algorithms are excluded from the considerations here.

This document represents the consensus of the Crypto Forum Research Group (CFRG).

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Basic Terms and Definitions

This document uses the following terms and definitions for the sets and operations on the elements of these sets:

V* the set of all bit strings of a finite length (hereinafter referred to as strings), including the empty string;

V_s the set of all bit strings of length s, where s is a non-negative integer;

|X| the bit length of the bit string X;

A | B the concatenation of strings A and B both belonging to V*, i.e., a string in V_{|A|+|B|}, where the left substring in V_|A| is equal to A and the right substring in V_|B| is equal to B;

(xor) the exclusive-or of two bit strings of the same length;

Z_{2^n} the ring of residues modulo 2^n;

Int_s: V_s -> Z_{2^s}
the transformation that maps the string a = (a_s, ... , a_1) in V_s into the integer Int_s(a) = 2^{s-1} * a_s + ... + 2 * a_2 + a_1 (the interpretation of the binary string as an integer);

Vec_s: Z_{2^s} -> V_s
the transformation inverse to the mapping Int_s (the interpretation of an integer as a binary string);

MSB_i: V_s -> V_i
the transformation that maps the string a = (a_s, ... , a_1) in V_s into the string MSB_i(a) = (a_s, ... , a_{s-i+1}) in V_i (most significant bits);

LSB_i: V_s -> V_i
the transformation that maps the string a = (a_s, ... , a_1) in V_s into the string LSB_i(a) = (a_i, ... , a_1) in V_i (least significant bits);

Inc_c: V_s -> V_s
the transformation that maps the string a = (a_s, ... , a_1) in V_s into the string Inc_c(a) = MSB_{|a|-c}(a) | Vec_c(Int_c(LSB_c(a)) + 1(mod 2^c)) in V_s (incrementing the least significant c bits of the bit string, regarded as the binary representation of an integer);

a^s the string in V_s that consists of s ’a’ bits;

E_{K}: V_n -> V_n
the block cipher permutation under the key K in V_k;

ceil(x) the smallest integer that is greater than or equal to x;

floor(x)
the biggest integer that is less than or equal to x;

k the bit length of the K; k is assumed to be divisible by 8;

n the block size of the block cipher (in bits); n is assumed to be divisible by 8;

b the number of data blocks in the plaintext P (b = ceil(|P|/n));

N the section size (the number of bits that are processed with one section key before this key is transformed).

A plaintext message P and the corresponding ciphertext C are divided into b = ceil(|P|/n) blocks, denoted as P = P_1 | P_2 | ... | P_b and C = C_1 | C_2 | ... | C_b, respectively. The first b-1 blocks P_i and C_i are in V_n for i = 1, 2, ... , b-1. The b-th blocks P_b and C_b may be incomplete blocks, i.e., in V_r, where r <= n if not otherwise specified.

4. Choosing Constructions and Security Parameters

External re-keying is an approach assuming that a key is transformed after encrypting a limited number of entire messages. The external re-keying method is chosen at the protocol level, regardless of the underlying block cipher or the encryption mode. External re-keying is recommended for protocols that process relatively short messages or protocols that have a way to divide a long message into manageable pieces. Through external re-keying, the number of messages that can be securely processed with a single initial key K is substantially increased without a loss of message length.

External re-keying has the following advantages

1. It increases the lifetime of an initial key by increasing the number of messages processed with this key.

2. It has minimal impact on performance when the number of messages processed under one initial key is sufficiently large.

3. It provides forward and backward security of data-processing keys.

However, the use of external re-keying has the following disadvantage: in cases with restrictive key lifetime limitations, the message sizes can become obstructive due to the impossibility of processing sufficiently large messages, so it may be necessary to perform additional fragmentation at the protocol level. For example, if the key lifetime L is 1 GB and the message length m = 3 GB, then this message cannot be processed as a whole, and it should be divided into three fragments that will be processed separately.

Internal re-keying is an approach assuming that a key is transformed during each separate message processing. Such procedures are integrated into the base modes of operations, so every internal re-keying mechanism is defined for the particular operation mode and the block size of the used cipher. Internal re-keying is recommended for protocols that process long messages: the size of each single message can be substantially increased without loss in the number of messages that can be securely processed with a single initial key.

Internal re-keying has the following advantages:

1. It increases the lifetime of an initial key by increasing the size of the messages processed with one initial key.

2. It has minimal impact on performance.

3. Internal re-keying mechanisms without a master key do not affect short-message transformation at all.

4. It is transparent (works like any mode of operation): it does not require changes of initialization vectors (IVs) and a restart of MACing.

However, the use of internal re-keying has the following disadvantages:

1. a specific method must not be chosen independently of a mode of operation.

2. internal re-keying mechanisms without a master key do not provide backward security of data-processing keys.

Any block cipher modes of operations with internal re-keying can be jointly used with any external re-keying mechanisms. Such joint usage increases both the number of messages processed with one initial key and their maximum possible size.

If the adversary has access to the data-processing interface, the use of the same cryptographic primitives both for data-processing and re-keying transformation decreases the code size but can lead to some possible vulnerabilities (the possibility of mounting a chosen-plaintext attack may lead to the compromise of the following keys). This vulnerability can be eliminated by using different primitives for data processing and re-keying, e.g., block cipher for data processing and hash for re-keying (see Section 5.2.2 and Section 5.3.2). However, in this case, the security of the whole scheme cannot be reduced to standard notions like a pseudorandom function (PRF) or pseudorandom permutation (PRP), so security estimations become more difficult and unclear.

Summing up the abovementioned issues briefly:

1. If a protocol assumes processing of long records (e.g., [CMS]), internal re-keying should be used. If a protocol assumes processing of a significant number of ordered records, which can be considered as a single data stream (e.g., [TLS], [SSH]), internal re-keying may also be used.

2. For protocols that allow out-of-order delivery and lost records (e.g., [DTLS], [ESP]), external re-keying should be used as, in this case, records cannot be considered as a single data stream. If the records are also long enough, internal re-keying should also be used during each separate message processing.

For external re-keying:

1. If it is desirable to separate transformations used for data processing and key updates, hash function-based re-keying should be used.

2. If parallel data processing is required, then parallel external re-keying should be used.

3. If restrictive key lifetime limitations are present, external tree-based re-keying should be used.

For internal re-keying:

1. If the property of forward and backward security is desirable for data-processing keys and if additional key material can be easily obtained for the data-processing stage, internal re-keying with a master key should be used.

5. External Re-keying Mechanisms

This section presents an approach to increasing the initial key lifetime by using a transformation of a data-processing key (frame key) after processing a limited number of entire messages (frame). The approach provides external parallel and serial re-keying mechanisms (see [AbBell]). These mechanisms use initial key K only for frame key generation and never use it directly for data processing. Such mechanisms operate outside of the base modes of operations and do not change them at all; therefore, they are called "external re-keying" mechanisms in this document.

External re-keying mechanisms are recommended for usage in protocols that process quite small messages, since the maximum gain in increasing the initial key lifetime is achieved by increasing the number of messages.

External re-keying increases the initial key lifetime through the following approach. Suppose there is a protocol P with some mode of operation (base encryption or authentication mode). Let L1 be a key lifetime limitation induced by side-channel analysis methods (side-channel limitation), let L2 be a key lifetime limitation induced by methods based on the combinatorial properties of a used mode of operation (combinatorial limitation), and let q1, q2 be the total numbers of messages of length m that can be safely processed with an initial key K according to these limitations.

Let L = min(L1, L2), q = min(q1, q2), and q * m <= L. As the L1 limitation is usually much stronger than the L2 limitation (L1 < L2), the final key lifetime restriction is equal to the most restrictive limitation L1. Thus, as displayed in Figure 2, without re-keying, only q1 (q1 * m <= L1) messages can be safely processed.

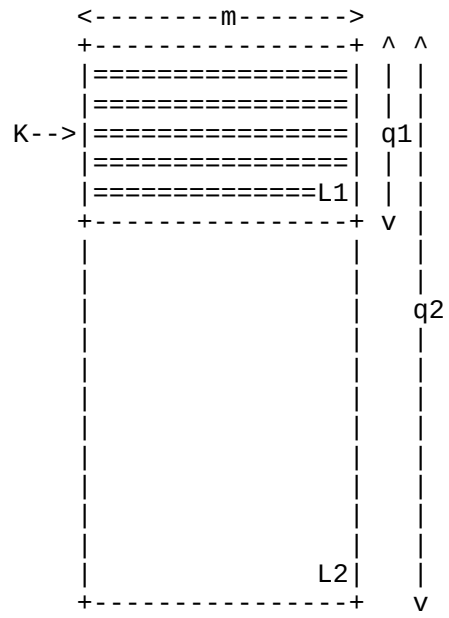


Figure 2: Basic Principles of Message Processing without External Re-keying

Suppose that the safety margin for the protocol P is fixed and the external re-keying approach is applied to the initial key K to generate the sequence of frame keys. The frame keys are generated in such a way that the leakage of a previous frame key does not have any impact on the following one, so the side-channel limitation L1 is switched off. Thus, the resulting key lifetime limitation of the initial key K can be calculated on the basis of a new combinatorial limitation L2'. It is proven (see [AbBell]) that the security of the mode of operation that uses external re-keying leads to an increase when compared to base mode without re-keying (thus, L2 < L2'). Hence, as displayed in Figure 3, the resulting key lifetime limitation if using external re-keying can be increased up to L2'.

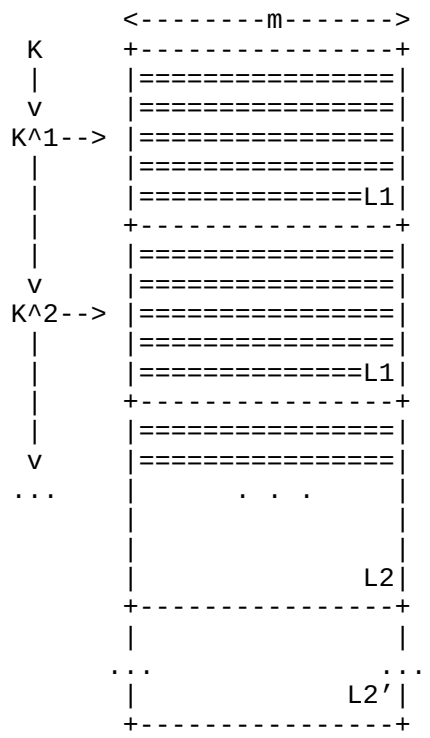


Figure 3: Basic Principles of Message Processing
with External Re-keying

Note: The key transformation process is depicted in a simplified form. A specific approach (parallel and serial) is described below.

Consider an example. Let the message size in a protocol P be equal to 1 KB. Suppose L1 = 128 MB and L2 = 1 TB. Thus, if an external re-keying mechanism is not used, the initial key K must be renegotiated after processing 128 MB / 1 KB = 131072 messages.

If an external re-keying mechanism is used, the key lifetime limitation L1 goes off. Hence, the resulting key lifetime limitation L2' can be set to more than 1 TB. Thus, if an external re-keying mechanism is used, more than 1 TB / 1 KB = 2^30 messages can be processed before the initial key K is renegotiated. This is 8192 times greater than the number of messages that can be processed when an external re-keying mechanism is not used.

5.1. Methods of Key Lifetime Control

Suppose L is an amount of data that can be safely processed with one frame key. For i in {1, 2, ... , t}, the frame key K^i (see Figures 4 and 6) should be transformed after processing q_i messages, where q_i can be calculated in accordance with one of the following approaches:

Explicit approach:

q_i is such that |M^{i,1}| + ... + |M^{i,q_i}| <= L, |M^{i,1}| + ... + |M^{i,q_i+1}| > L.
This approach allows use of the frame key K^i in an almost optimal way, but it can be applied only when messages cannot be lost or reordered (e.g., TLS records).

Implicit approach:

q_i = L / m_max, i = 1, ... , t.
The amount of data processed with one frame key K^i is calculated under the assumption that every message has the maximum length m_max. Hence, this amount can be considerably less than the key lifetime limitation L. On the other hand, this approach can be applied when messages may be lost or reordered (e.g., DTLS records).

Dynamic key changes:

We can organize the key change using the Protected Point to Point ([P3]) solution by building a protected tunnel between the endpoints in which the information about frame key updating can be safely passed across. This can be useful, for example, when we want the adversary to not detect the key change during the protocol evaluation.

5.2. Parallel Constructions

External parallel re-keying mechanisms generate frame keys K^1, K^2, ... directly from the initial key K independently of each other.

The main idea behind external re-keying with a parallel construction is presented in Figure 4:

Maximum message size = m_max.

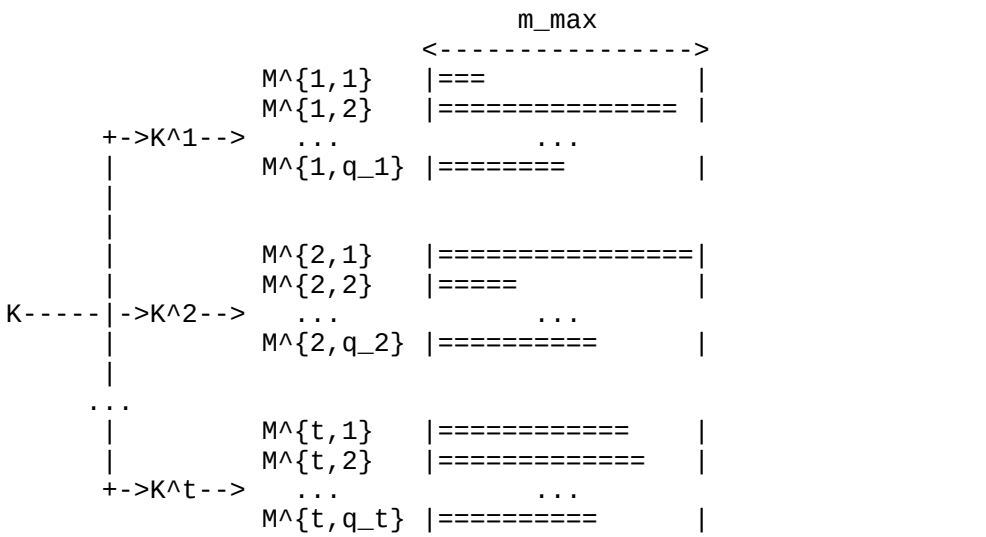


Figure 4: External Parallel Re-keying Mechanisms

The frame key K^i, i = 1, ... , t - 1 is updated after processing a certain number of messages (see Section 5.1).

5.2.1. Parallel Construction Based on a KDF on a Block Cipher

The ExtParallelC re-keying mechanism is based on the key derivation function on a block cipher and is used to generate t frame keys as follows:

K^1 | K^2 | ... | K^t = ExtParallelC(K, t * k) = MSB_{t * k}(E_{K}(Vec_n(0)) | E_{K}(Vec_n(1)) | ... | E_{K}(Vec_n(R - 1))),

where R = ceil(t * k/n).

5.2.2. Parallel Construction Based on a KDF on a Hash Function

The ExtParallelH re-keying mechanism is based on the key derivation function HKDF-Expand, described in [RFC5869], and is used to generate t frame keys as follows:

```
K^1 | K^2 | ... | K^t = ExtParallelH(K, t * k) = HKDF-Expand(K,
label, t * k),
```

where label is a string (may be a zero-length string) that is defined by a specific protocol.

5.2.3. Tree-Based Construction

The application of an external tree-based mechanism leads to the construction of the key tree with the initial key K (root key) at the 0 level and the frame keys K^1, K^2, ... at the last level, as described in Figure 5.

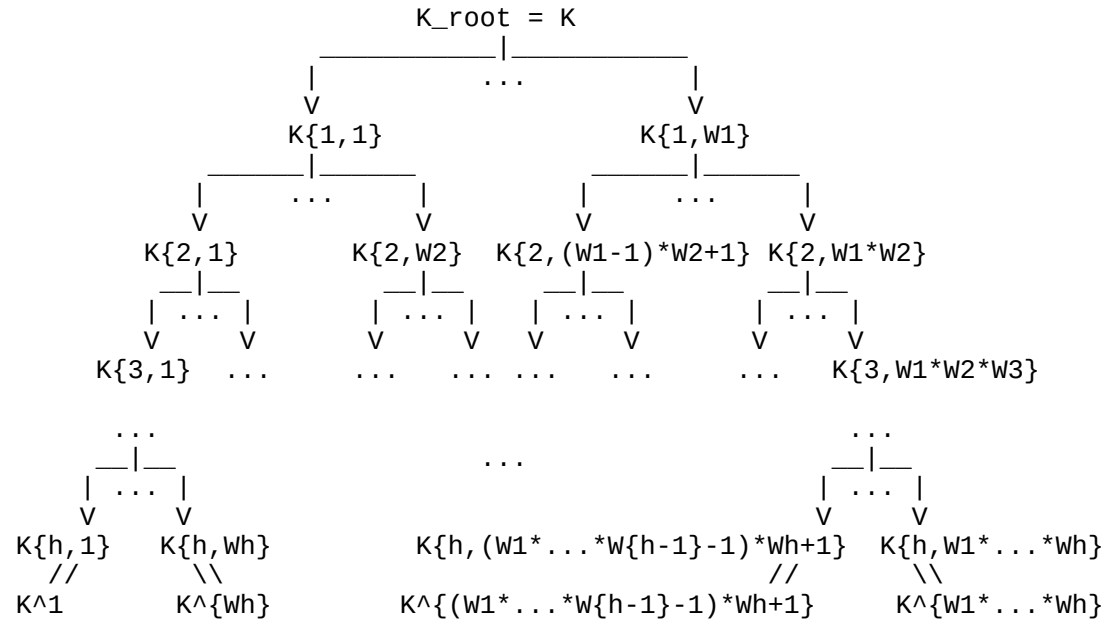


Figure 5: External Tree-Based Mechanism

The tree height h and the number of keys Wj, j in {1, ... , h}, which can be partitioned from the "parent" key, are defined in accordance with a specific protocol and key lifetime limitations for the used derivation functions.

Each j-level key K{j,w}, where j in {1, ... , h}, w in {1, ... , W1 * ... * Wj}, is derived from the (j-1)-level "parent" key K{j-1, ceil(w/Wi)} (and other appropriate input data) using the j-th level derivation function. This function can be based on the block cipher function or on the hash function and is defined in accordance with a specific protocol.

The i-th frame K^i, i in {1, 2, ... , W1*...*Wh}, can be calculated as follows:

K^i = ExtKeyTree(K, i) = KDF_h(KDF_{h-1}(... KDF_1(K, ceil(i / (W2 * ... * Wh)) ... , ceil(i / Wh)), i),

where KDF_j is the j-th level derivation function that takes two arguments (the parent key value and the integer in a range from 1 to W1 * ... * Wj) and outputs the j-th level key value.

The frame key K^i is updated after processing a certain number of messages (see Section 5.1).

In order to create an efficient implementation, during frame key K^i generation, the derivation functions KDF_j, j in {1, ... , h-1} should be used only when ceil(i / (W{j+1} * ... * Wh)) != ceil((i - 1) / (W{j+1} * ... * Wh)); otherwise, it is necessary to use a previously generated value. This approach also makes it possible to take countermeasures against side-channel attacks.

Consider an example. Suppose h = 3, W1 = W2 = W3 = W, and KDF_1, KDF_2, KDF_3 are key derivation functions based on the KDF_GOSTR3411_2012_256 (hereafter simply KDF) function described in [RFC7836]. The resulting ExtKeyTree function can be defined as follows:

ExtKeyTree(K, i) = KDF(KDF(KDF(K, "level1", ceil(i / W^2)), "level2", ceil(i / W)), "level3", i).

where i in {1, 2, ... , W^3}.

A structure similar to the external tree-based mechanism can be found in Section 6 of [NISTSP800-108].

5.3. Serial Constructions

External serial re-keying mechanisms generate frame keys, each of which depends on the secret state (K*_1, K*_2, ...) that is updated after the generation of each new frame key; see Figure 6. Similar approaches are used in the [SIGNAL] protocol and the [TLS] updating

traffic key mechanism and were proposed for use in the [U2F] protocol.

External serial re-keying mechanisms have the obvious disadvantage of being impossible to implement in parallel, but they may be the preferred option if additional forward secrecy is desirable. If all keys are securely deleted after usage, the compromise of a current secret state at some point does not lead to a compromise of all previous secret states and frame keys. In terms of [TLS], compromise of application_traffic_secret_N does not compromise all previous application_traffic_secret_i, i < N.

The main idea behind external re-keying with a serial construction is presented in Figure 6:

Maximum message size = m_max.

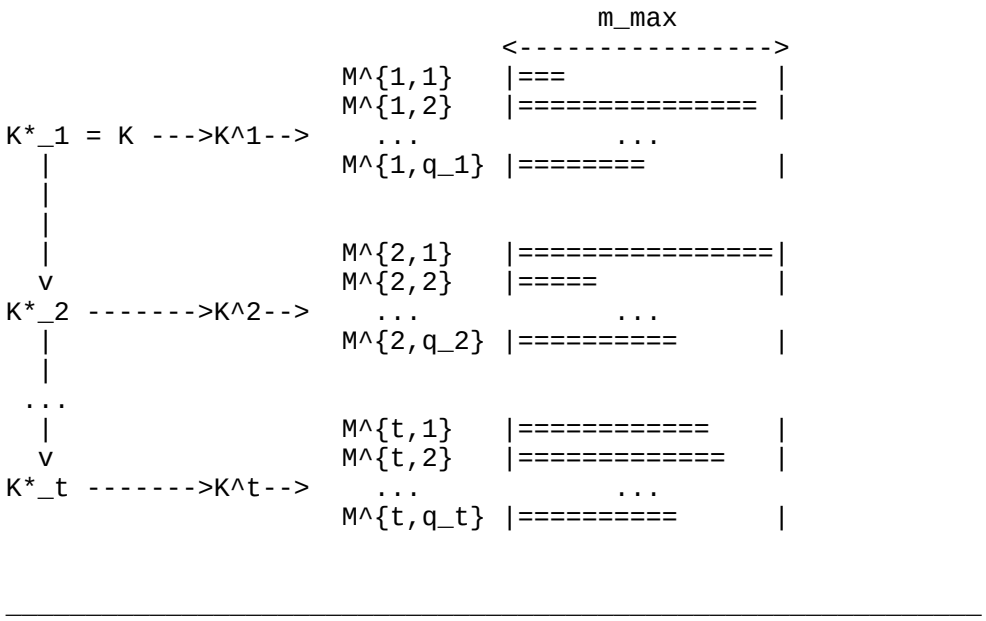


Figure 6: External Serial Re-keying Mechanisms

The frame key K^i, i = 1, ... , t - 1, is updated after processing a certain number of messages (see Section 5.1).

5.3.1. Serial Construction Based on a KDF on a Block Cipher

The frame key K^i is calculated using the ExtSerialC transformation as follows:

```
K^i = ExtSerialC(K, i) =
MSB_k(E_{K*_i}(Vec_n(0)) |E_{K*_i}(Vec_n(1)) | ... |
E_{K*_i}(Vec_n(J - 1))),
```

where J = ceil(k / n), i = 1, ... , t, K*_i is calculated as follows:

```
K*_1 = K,

K*_{j+1} = MSB_k(E_{K*_j}(Vec_n(J)) | E_{K*_j}(Vec_n(J + 1)) |
... |
E_{K*_j}(Vec_n(2 * J - 1))),
```

where j = 1, ... , t - 1.

5.3.2. Serial Construction Based on a KDF on a Hash Function

The frame key K^i is calculated using the ExtSerialH transformation as follows:

```
K^i = ExtSerialH(K, i) = HKDF-Expand(K*_i, label1, k),
```

where i = 1, ... , t; HKDF-Expand is the HMAC-based key derivation function, as described in [RFC5869]; and K*_i is calculated as follows:

```
K*_1 = K,

K*_{j+1} = HKDF-Expand(K*_j, label2, k), where j = 1, ... , t - 1,
```

where label1 and label2 are different strings from V* that are defined by a specific protocol (see, for example, the algorithm for updating traffic keys in TLS 1.3 [TLS]).

5.4. Using Additional Entropy during Re-keying

In many cases, using additional entropy during re-keying won't increase security but may give a false sense of that. Therefore, one can rely on additional entropy only after conducting a deep security analysis. For example, good PRF constructions do not require additional entropy for the quality of keys, so, in most cases, there is no need to use additional entropy with external re-keying mechanisms based on secure KDFs. However, in some situations, mixed-in entropy can still increase security in the case of a time-limited

but complete breach of the system when an adversary can access the frame-key generation interface but cannot reveal the master keys (e.g., when the master keys are stored in a Hardware Security Module (HSM)).

For example, an external parallel construction based on a KDF on a hash function with a mixed-in entropy can be described as follows:

```
K^i = HKDF-Expand(K, label_i, k),
```

where label_i is additional entropy that must be sent to the recipient (e.g., sent jointly with an encrypted message). The entropy label_i and the corresponding key K^i must be generated directly before message processing.

6. Internal Re-keying Mechanisms

This section presents an approach to increasing the key lifetime by using a transformation of a data-processing key (section key) during each separate message processing. Each message is processed starting with the same key (the first section key), and each section key is updated after processing N bits of the message (section).

This section provides internal re-keying mechanisms called ACPKM (Advanced Cryptographic Prolongation of Key Material) and ACPKM-Master that do not use a master key and use a master key, respectively. Such mechanisms are integrated into the base modes of operation and actually form new modes of operation. Therefore, they are called "internal re-keying" mechanisms in this document.

Internal re-keying mechanisms are recommended to be used in protocols that process large single messages (e.g., CMS messages), since the maximum gain in increasing the key lifetime is achieved by increasing the length of a message, while it provides almost no increase in the number of messages that can be processed with one initial key.

Internal re-keying increases the key lifetime through the following approach. Suppose protocol P uses some base mode of operation. Let L1 and L2 be a side channel and combinatorial limitations, respectively, and for some fixed number of messages q, let m1, m2 be the lengths of messages that can be safely processed with a single initial key K according to these limitations.

Thus, the approach without re-keying (analogous to Section 5) yields a final key lifetime restriction equal to L1, and only q messages of the length m1 can be safely processed; see Figure 7.

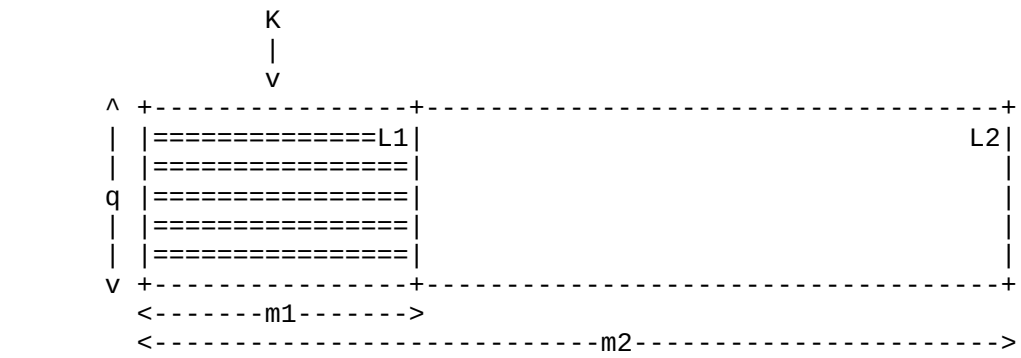


Figure 7: Basic Principles of Message Processing without Internal Re-keying

Suppose that the safety margin for the protocol P is fixed and the internal re-keying approach is applied to the base mode of operation. Suppose further that every message is processed with a section key, which is transformed after processing N bits of data, where N is a parameter. If q * N does not exceed L1, then the side-channel limitation L1 goes off, and the resulting key lifetime limitation of the initial key K can be calculated on the basis of a new combinatorial limitation L2'. The security of the mode of operation that uses internal re-keying increases when compared to the base mode of operation without re-keying (thus, L2 < L2'). Hence, as displayed in Figure 8, the resulting key lifetime limitation if using internal re-keying can be increased up to L2'.

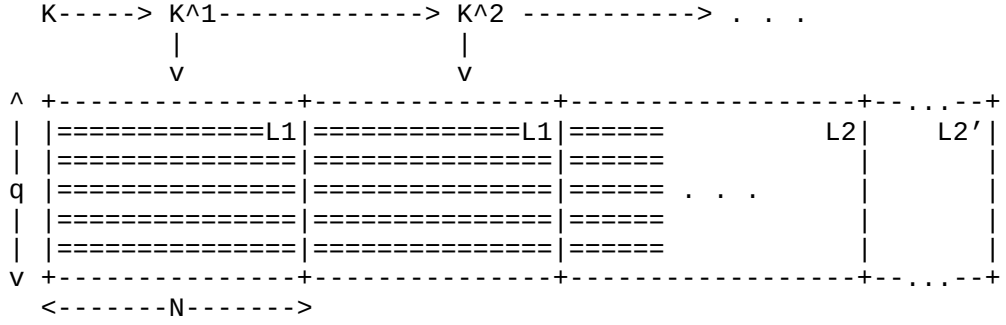


Figure 8: Basic Principles of Message Processing with Internal Re-keying

Note: The key transformation process is depicted in a simplified form. A specific approach (ACPKM and ACPKM-Master re-keying mechanisms) is described below.

Since the performance of encryption can slightly decrease for rather small values of N, the maximum possible value should be selected for parameter N for a particular protocol in order to provide the necessary key lifetime for the considered security models.

Consider an example. Suppose L1 = 128 MB and L2 = 10 TB. Let the message size in the protocol be large/unlimited (which may exhaust the whole key lifetime L2). The most restrictive resulting key lifetime limitation is equal to 128 MB.

Thus, there is a need to put a limit on the maximum message size m_max. For example, if m_max = 32 MB, it may happen that the renegotiation of initial key K would be required after processing only four messages.

If an internal re-keying mechanism with section size N = 1 MB is used, more than L1 / N = 128 MB / 1 MB = 128 messages can be processed before the renegotiation of initial key K (instead of four messages when an internal re-keying mechanism is not used). Note that only one section of each message is processed with the section key K^i, and, consequently, the key lifetime limitation L1 goes off. Hence, the resulting key lifetime limitation L2’ can be set to more than 10 TB (in cases when a single large message is processed using the initial key K).

6.1. Methods of Key Lifetime Control

Suppose L is an amount of data that can be safely processed with one section key and N is a section size (fixed parameter). Suppose M^{i}_1 is the first section of message M^{i}, i = 1, ... , q (see Figures 9 and 10); the parameter q can then be calculated in accordance with one of the following two approaches:

- Explicit approach:
  q_i is such that |M^{1}_1| + ... + |M^{q}_1| <= L, |M^{1}_1| + ... + |M^{q+1}_1| > L
  This approach allows use of the section key K^i in an almost optimal way, but it can be applied only when messages cannot be lost or reordered (e.g., TLS records).

- Implicit approach:
  q = L / N.
  The amount of data processed with one section key K^i is calculated under the assumption that the length of every message is equal to or greater than section size N and thus can be considerably less than the key lifetime limitation L. On the other hand, this approach can be applied when messages may be lost or reordered (e.g., DTLS records).

6.2. Constructions that Do Not Require a Master Key

This section describes the block cipher modes that use the ACPKM re-keying mechanism, which does not use a master key; an initial key is used directly for the data encryption.

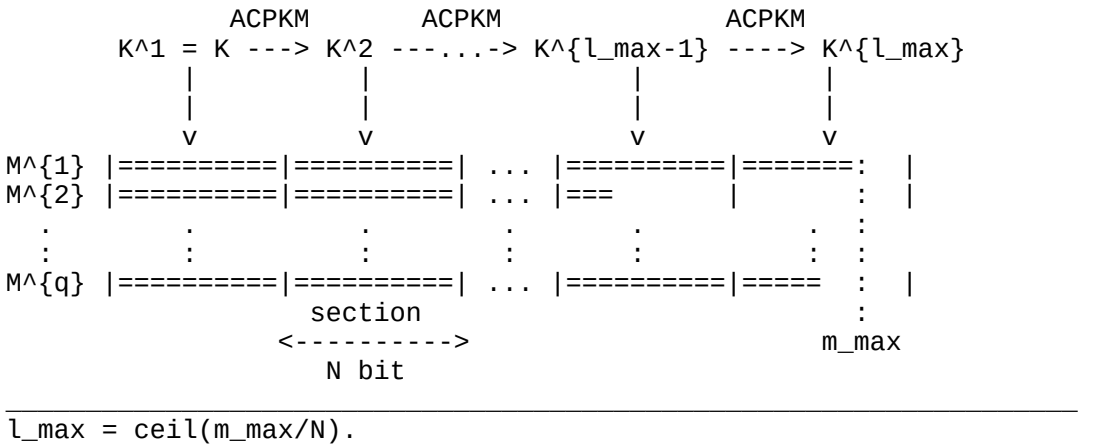
6.2.1. ACPKM Re-keying Mechanisms

This section defines a periodical key transformation without a master key, which is called the ACPKM re-keying mechanism. This mechanism can be applied to one of the base encryption modes (CTR and GCM block cipher modes) to get an extension of this encryption mode that uses periodical key transformation without a master key. This extension can be considered as a new encryption mode.

An additional parameter that defines the functioning of base encryption modes with the ACPKM re-keying mechanism is the section size N. The value of N is measured in bits and is fixed within a specific protocol based on the requirements of the system capacity and the key lifetime. The section size N MUST be divisible by the block size n.

The main idea behind internal re-keying without a master key is presented in Figure 9:

Section size = const = N,
maximum message size = m_max.



l_max = ceil(m_max/N).

Figure 9: Internal Re-keying without a Master Key

During the processing of the input message M with the length m in some encryption mode that uses the ACPKM key transformation of the initial key K, the message is divided into l = ceil(m / N) sections (denoted as M = M_1 | M_2 | ... | M_l, where M_i is in V_N for i in {1, 2, ... , l - 1} and M_l is in V_r, r <= N). The first section of each message is processed with the section key K^1 = K. To process the (i + 1)-th section of each message, the section key K^{i+1} is calculated using the ACPKM transformation as follows:

$$K^{i+1} = ACPKM(K^i) = MSB_k(E_{K^i}(D_1) \mid ... \mid E_{K^i}(D_J)),$$

where J = ceil(k/n) and D_1, D_2, ... , D_J are in V_n and are calculated as follows:

$$D_1 \mid D_2 \mid ... \mid D_J = MSB_{J * n}(D),$$

where D is the following constant in V_{1024}:

```
D = ( 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87
    | 88 | 89 | 8a | 8b | 8c | 8d | 8e | 8f
    | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97
    | 98 | 99 | 9a | 9b | 9c | 9d | 9e | 9f
    | a0 | a1 | a2 | a3 | a4 | a5 | a6 | a7
    | a8 | a9 | aa | ab | ac | ad | ae | af
    | b0 | b1 | b2 | b3 | b4 | b5 | b6 | b7
    | b8 | b9 | ba | bb | bc | bd | be | bf
    | c0 | c1 | c2 | c3 | c4 | c5 | c6 | c7
    | c8 | c9 | ca | cb | cc | cd | ce | cf
    | d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7
    | d8 | d9 | da | db | dc | dd | de | df
    | e0 | e1 | e2 | e3 | e4 | e5 | e6 | e7
    | e8 | e9 | ea | eb | ec | ed | ee | ef
    | f0 | f1 | f2 | f3 | f4 | f5 | f6 | f7
    | f8 | f9 | fa | fb | fc | fd | fe | ff)
```

Note: The constant D is such that D_1, ... , D_J are pairwise different for any allowed n and k values.

Note: The highest bit of each octet of the constant D is equal to 1. This condition is important as, in conjunction with a certain mode message length limitation, it allows prevention of collisions of block cipher permutation inputs in cases with key transformation and message processing (for more details, see Section 4.4 of [AAOS2017]).

6.2.2. CTR-ACPKM Encryption Mode

This section defines a CTR-ACPKM encryption mode that uses the ACPKM internal re-keying mechanism for the periodical key transformation.

The CTR-ACPKM mode can be considered as the base encryption mode CTR (see [MODES]) extended by the ACPKM re-keying mechanism.

The CTR-ACPKM encryption mode can be used with the following parameters:

- 64 <= n <= 512.
- 128 <= k <= 512.
- The number c of bits in a specific part of the block to be incremented is such that 32 <= c <= 3 / 4 n, where c is a multiple of 8.
- The maximum message size m_max = n * 2^{c-1}.

The CTR-ACPKM mode encryption and decryption procedures are defined as follows:

```
+-------------------------------------------------------------------+
|  CTR-ACPKM-Encrypt(N, K, ICN, P)                                  |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - initial counter nonce ICN in V_{n-c},                          |
|  - plaintext P = P_1 | ... | P_b, |P| <= m_max.                   |
|  Output:                                                          |
|  - ciphertext C.                                                  |
|-------------------------------------------------------------------|
|  1. CTR_1 = ICN | 0^c                                             |
|  2. For j = 2, 3, ... , b do                                      |
|         CTR_{j} = Inc_c(CTR_{j-1})                                |
|  3. K^1 = K                                                       |
|  4. For i = 2, 3, ... , ceil(|P| / N)                             |
|         K^i = ACPKM(K^{i-1})                                      |
|  5. For j = 1, 2, ... , b do                                      |
|         i = ceil(j * n / N),                                      |
|         G_j = E_{K^i}(CTR_j)                                      |
|  6. C = P (xor) MSB_{|P|}(G_1 | ... | G_b)                        |
|  7. Return C                                                      |
+-------------------------------------------------------------------+
```

```
+-------------------------------------------------------------+
|  CTR-ACPKM-Decrypt(N, K, ICN, C)                            |
|-------------------------------------------------------------|
|  Input:                                                     |
|  - section size N,                                          |
|  - initial key K,                                           |
|  - initial counter nonce ICN in V_{n-c},                    |
|  - ciphertext C = C_1 | ... | C_b, |C| <= m_max.           |
|  Output:                                                    |
|  - plaintext P.                                             |
|-------------------------------------------------------------|
|  1. P = CTR-ACPKM-Encrypt(N, K, ICN, C)                     |
|  2. Return P                                                |
+-------------------------------------------------------------+
```

The initial counter nonce (ICN) value for each message that is encrypted under the given initial key K must be chosen in a unique manner.

6.2.3. GCM-ACPKM Authenticated Encryption Mode

This section defines the GCM-ACPKM authenticated encryption mode that uses the ACPKM internal re-keying mechanism for the periodical key transformation.

The GCM-ACPKM mode can be considered as the base authenticated encryption mode GCM (see [GCM]) extended by the ACPKM re-keying mechanism.

The GCM-ACPKM authenticated encryption mode can be used with the following parameters:

- n in {128, 256}.
- 128 <= k <= 512.
- The number c of bits in a specific part of the block to be incremented is such that 1 / 4 n <= c <= 1 / 2 n, c is a multiple of 8.
- Authentication tag length t.
- The maximum message size m_max = min{n * (2^{c-1} - 2), 2^{n/2} - 1}.

The GCM-ACPKM mode encryption and decryption procedures are defined as follows:

```
+-------------------------------------------------------------------+
|  GHASH(X, H)                                                      |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - bit string X = X_1 | ... | X_m, X_1, ... , X_m in V_n.         |
|  Output:                                                          |
|  - block GHASH(X, H) in V_n.                                      |
|-------------------------------------------------------------------|
|  1. Y_0 = 0^n                                                     |
|  2. For i = 1, ... , m do                                         |
|         Y_i = (Y_{i-1} (xor) X_i) * H                             |
|  3. Return Y_m                                                    |
+-------------------------------------------------------------------+
```

```
+-------------------------------------------------------------------+
|  GCTR(N, K, ICB, X)                                               |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - initial counter block ICB,                                     |
|  - X = X_1 | ... | X_b.                                           |
|  Output:                                                          |
|  - Y in V_{|X|}.                                                  |
|-------------------------------------------------------------------|
|  1. If X in V_0, then return Y, where Y in V_0                    |
|  2. GCTR_1 = ICB                                                  |
|  3. For i = 2, ... , b do                                         |
|         GCTR_i = Inc_c(GCTR_{i-1})                                |
|  4. K^1 = K                                                       |
|  5. For j = 2, ... , ceil(|X| / N)                                |
|         K^j = ACPKM(K^{j-1})                                      |
|  6. For i = 1, ... , b do                                         |
|         j = ceil(i * n / N),                                      |
|         G_i = E_{K_j}(GCTR_i)                                     |
|  7. Y = X (xor) MSB_{|X|}(G_1 | ... | G_b)                       |
|  8. Return Y                                                      |
+-------------------------------------------------------------------+
```

```
+-------------------------------------------------------------------+
|  GCM-ACPKM-Encrypt(N, K, ICN, P, A)                               |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - initial counter nonce ICN in V_{n-c},                          |
|  - plaintext P = P_1 | ... | P_b, |P| <= m_max,                   |
|  - additional authenticated data A.                               |
|  Output:                                                          |
|  - ciphertext C,                                                  |
|  - authentication tag T.                                          |
|-------------------------------------------------------------------|
|  1. H = E_{K}(0^n)                                                |
|  2. ICB_0 = ICN | 0^{c-1} | 1                                     |
|  3. C = GCTR(N, K, Inc_c(ICB_0), P)                               |
|  4. u = n * ceil(|C| / n) - |C|                                   |
|     v = n * ceil(|A| / n) - |A|                                   |
|  5. S = GHASH(A | 0^v | C | 0^u | Vec_{n/2}(|A|) |               |
|               | Vec_{n/2}(|C|), H)                              |
|  6. T = MSB_t(E_{K}(ICB_0) (xor) S)                               |
|  7. Return C | T                                                  |
+-------------------------------------------------------------------+

+-------------------------------------------------------------------+
|  GCM-ACPKM-Decrypt(N, K, ICN, A, C, T)                            |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - initial counter block ICN,                                     |
|  - additional authenticated data A,                               |
|  - ciphertext C = C_1 | ... | C_b, |C| <= m_max,                 |
|  - authentication tag T.                                          |
|  Output:                                                          |
|  - plaintext P or FAIL.                                           |
|-------------------------------------------------------------------|
|  1. H = E_{K}(0^n)                                                |
|  2. ICB_0 = ICN | 0^{c-1} | 1                                     |
|  3. P = GCTR(N, K, Inc_c(ICB_0), C)                               |
|  4. u = n * ceil(|C| / n) - |C|                                   |
|     v = n * ceil(|A| / n) - |A|                                   |
|  5. S = GHASH(A | 0^v | C | 0^u | Vec_{n/2}(|A|) |               |
|               | Vec_{n/2}(|C|), H)                              |
|  6. T' = MSB_t(E_{K}(ICB_0) (xor) S)                              |
|  7. If T = T', then return P; else return FAIL                    |
+-------------------------------------------------------------------+
```

The * operation on (pairs of) the 2^n possible blocks corresponds to the multiplication operation for the binary Galois (finite) field of 2^n elements defined by the polynomial f as follows (analogous to [GCM]):

n = 128: f = a^128 + a^7 + a^2 + a^1 + 1,

n = 256: f = a^256 + a^10 + a^5 + a^2 + 1.

The initial counter nonce ICN value for each message that is encrypted under the given initial key K must be chosen in a unique manner.

The key for computing values E_{K}(ICB_0) and H is not updated and is equal to the initial key K.

6.3. Constructions that Require a Master Key

This section describes the block cipher modes that use the ACPKM-Master re-keying mechanism, which use the initial key K as a master key, so K is never used directly for data processing but is used for key derivation.

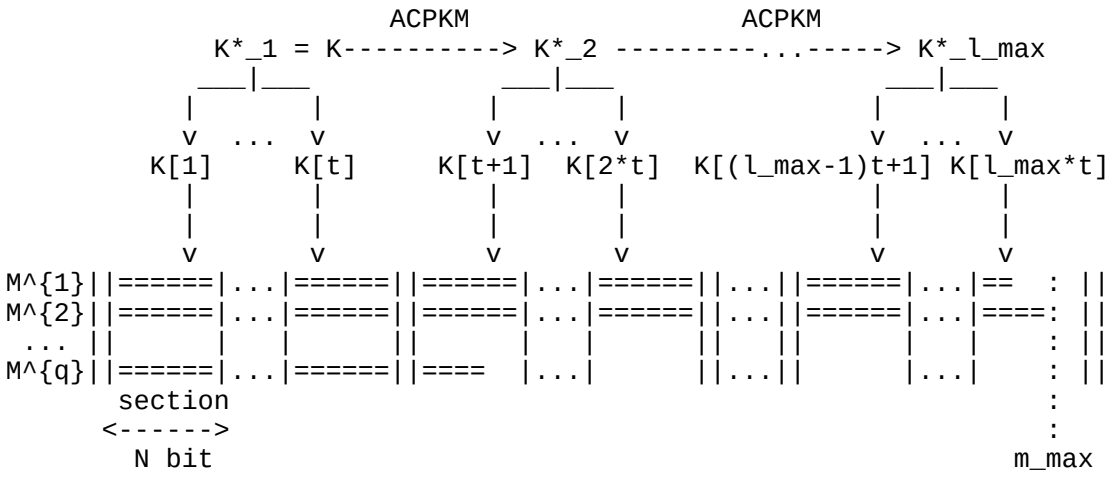
6.3.1. ACPKM-Master Key Derivation from the Master Key

This section defines periodical key transformation with a master key, which is called the ACPKM-Master re-keying mechanism. This mechanism can be applied to one of the base modes of operation (CTR, GCM, CBC, CFB, OMAC modes) for getting an extension that uses periodical key transformation with a master key. This extension can be considered as a new mode of operation.

Additional parameters that define the functioning of modes of operation that use the ACPKM-Master re-keying mechanism are the section size N, the change frequency T* of the master keys K*_1, K*_2, ... (see Figure 10), and the size d of the section key material. The values of N and T* are measured in bits and are fixed within a specific protocol based on the requirements of the system capacity and the key lifetime. The section size N MUST be divisible by the block size n. The master key frequency T* MUST be divisible by d and by n.

The main idea behind internal re-keying with a master key is presented in Figure 10:

Master key frequency T*,
section size N,
maximum message size = m_max.



|K[i]| = d,
t = T* / d,
l_max = ceil(m_max / (N * t)).

Figure 10: Internal Re-keying with a Master Key

During the processing of the input message M with the length m in some mode of operation that uses ACPKM-Master key transformation with the initial key K and the master key frequency T*, the message M is divided into l = ceil(m / N) sections (denoted as M = M_1 | M_2 | ... | M_l, where M_i is in V_N for i in {1, 2, ... , l - 1} and M_l is in V_r, r <= N). The j-th section of each message is processed with the key material K[j], j in {1, ... , l}, |K[j]| = d, which is calculated with the ACPKM-Master algorithm as follows:

K[1] | ... | K[l] = ACPKM-Master(T*, K, d, l) = CTR-ACPKM-Encrypt (T*, K, 1^{n/2}, 0^{d*l}).

Note: The parameters d and l MUST be such that d * l <= n * 2^{n/2-1}.

6.3.2. CTR-ACPKM-Master Encryption Mode

This section defines a CTR-ACPKM-Master encryption mode that uses the ACPKM-Master internal re-keying mechanism for the periodical key transformation.

The CTR-ACPKM-Master encryption mode can be considered as the base encryption mode CTR (see [MODES]) extended by the ACPKM-Master re-keying mechanism.

The CTR-ACPKM-Master encryption mode can be used with the following parameters:

- 64 <= n <= 512.
- 128 <= k <= 512.
- The number c of bits in a specific part of the block to be incremented is such that 32 <= c <= 3 / 4 n, c is a multiple of 8.
- The maximum message size m_max = min{N * (n * 2^{n/2-1} / k), n * 2^c}.

The key material K[j] that is used for one-section processing is equal to K^j, where |K^j| = k bits.

The CTR-ACPKM-Master mode encryption and decryption procedures are defined as follows:

```
+-------------------------------------------------------------------+
|  CTR-ACPKM-Master-Encrypt(N, K, T*, ICN, P)                       |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initial counter nonce ICN in V_{n-c},                          |
|  - plaintext P = P_1 | ... | P_b, |P| <= m_max.                   |
|  Output:                                                          |
|  - ciphertext C.                                                  |
|-------------------------------------------------------------------|
|  1. CTR_1 = ICN | 0^c                                             |
|  2. For j = 2, 3, ... , b do                                      |
|         CTR_{j} = Inc_c(CTR_{j-1})                                |
|  3. l = ceil(|P| / N)                                             |
|  4. K^1 | ... | K^l = ACPKM-Master(T*, K, k, l)                   |
|  5. For j = 1, 2, ... , b do                                      |
|         i = ceil(j * n / N),                                      |
|         G_j = E_{K^i}(CTR_j)                                      |
|  6. C = P (xor) MSB_{|P|}(G_1 | ... |G_b)                         |
|  7. Return C                                                      |
+-------------------------------------------------------------------+

+-------------------------------------------------------------------+
|  CTR-ACPKM-Master-Decrypt(N, K, T*, ICN, C)                       |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initial counter nonce ICN in V_{n-c},                          |
|  - ciphertext C = C_1 | ... | C_b, |C| <= m_max.                   |
|  Output:                                                          |
|  - plaintext P.                                                   |
|-------------------------------------------------------------------|
|  1. P = CTR-ACPKM-Master-Encrypt(N, K, T*, ICN, C)                |
|  1. Return P                                                      |
+-------------------------------------------------------------------+
```

The initial counter nonce ICN value for each message that is encrypted under the given initial key must be chosen in a unique manner.

6.3.3. GCM-ACPKM-Master Authenticated Encryption Mode

This section defines a GCM-ACPKM-Master authenticated encryption mode that uses the ACPKM-Master internal re-keying mechanism for the periodical key transformation.

The GCM-ACPKM-Master authenticated encryption mode can be considered as the base authenticated encryption mode GCM (see [GCM]) extended by the ACPKM-Master re-keying mechanism.

The GCM-ACPKM-Master authenticated encryption mode can be used with the following parameters:

- n in {128, 256}.
- 128 <= k <= 512.
- The number c of bits in a specific part of the block to be incremented is such that 1 / 4 n <= c <= 1 / 2 n, c is a multiple of 8.
- authentication tag length t.
- the maximum message size m_max = min{N * (n * 2^{n/2-1} / k), n * (2^c - 2), 2^{n/2} - 1}.

The key material K[j] that is used for the j-th section processing is equal to K^j, |K^j| = k bits.

The GCM-ACPKM-Master mode encryption and decryption procedures are defined as follows:

```
+-------------------------------------------------------------------+
|  GHASH(X, H)                                                      |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - bit string X = X_1 | ... | X_m, X_i in V_n for i in {1, ... ,m}|
|  Output:                                                          |
|  - block GHASH(X, H) in V_n                                       |
|-------------------------------------------------------------------|
|  1. Y_0 = 0^n                                                     |
|  2. For i = 1, ... , m do                                         |
|         Y_i = (Y_{i-1} (xor) X_i) * H                             |
|  3. Return Y_m                                                    |
+-------------------------------------------------------------------+
```

```
+-------------------------------------------------------------------+
|  GCTR(N, K, T*, ICB, X)                                           |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initial counter block ICB,                                     |
|  - X = X_1 | ... | X_b.                                           |
|  Output:                                                          |
|  - Y in V_{|X|}.                                                  |
|-------------------------------------------------------------------|
|  1. If X in V_0, then return Y, where Y in V_0                    |
|  2. GCTR_1 = ICB                                                  |
|  3. For i = 2, ... , b do                                         |
|         GCTR_i = Inc_c(GCTR_{i-1})                                |
|  4. l = ceil(|X| / N)                                             |
|  5. K^1 | ... | K^l = ACPKM-Master(T*, K, k, l)                   |
|  6. For j = 1, ... , b do                                         |
|         i = ceil(j * n / N),                                      |
|         G_j = E_{K^i}(GCTR_j)                                     |
|  7. Y = X (xor) MSB_{|X|}(G_1 | ... | G_b)                       |
|  8. Return Y                                                      |
+-------------------------------------------------------------------+
```

```
+-------------------------------------------------------------------+
|  GCM-ACPKM-Master-Encrypt(N, K, T*, ICN, P, A)                    |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initial counter nonce ICN in V_{n-c},                          |
|  - plaintext P = P_1 | ... | P_b, |P| <= m_max.                   |
|  - additional authenticated data A.                               |
|  Output:                                                          |
|  - ciphertext C,                                                  |
|  - authentication tag T.                                          |
|-------------------------------------------------------------------|
|  1. K^1 = ACPKM-Master(T*, K, k, 1)                               |
|  2. H = E_{K^1}(0^n)                                              |
|  3. ICB_0 = ICN | 0^{c-1} | 1                                     |
|  4. C = GCTR(N, K, T*, Inc_c(ICB_0), P)                           |
|  5. u = n * ceil(|C| / n) - |C|                                   |
|     v = n * ceil(|A| / n) - |A|                                   |
|  6. S = GHASH(A | 0^v | C | 0^u | Vec_{n/2}(|A|) |               |
|              | Vec_{n/2}(|C|), H)                                 |
|  7. T = MSB_t(E_{K^1}(ICB_0) (xor) S)                             |
|  8. Return C | T                                                  |
+-------------------------------------------------------------------+
```

```
+-------------------------------------------------------------------+
|  GCM-ACPKM-Master-Decrypt(N, K, T*, ICN, A, C, T)                 |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initial counter nonce ICN in V_{n-c},                          |
|  - additional authenticated data A.                               |
|  - ciphertext C = C_1 | ... | C_b, |C| <= m_max,                 |
|  - authentication tag T.                                          |
|  Output:                                                          |
|  - plaintext P or FAIL.                                           |
|-------------------------------------------------------------------|
|  1. K^1 = ACPKM-Master(T*, K, k, 1)                               |
|  2. H = E_{K^1}(0^n)                                              |
|  3. ICB_0 = ICN | 0^{c-1} | 1                                     |
|  4. P = GCTR(N, K, T*, Inc_c(ICB_0), C)                           |
|  5. u = n * ceil(|C| / n) - |C|                                   |
|     v = n * ceil(|A| / n) - |A|                                   |
|  6. S = GHASH(A | 0^v | C | 0^u | Vec_{n/2}(|A|) |               |
|               | Vec_{n/2}(|C|), H)                              |
|  7. T' = MSB_t(E_{K^1}(ICB_0) (xor) S)                            |
|  8. If T = T', then return P; else return FAIL.                   |
+-------------------------------------------------------------------+
```

The * operation on (pairs of) the 2^n possible blocks corresponds to the multiplication operation for the binary Galois (finite) field of 2^n elements defined by the polynomial f as follows (by analogy with [GCM]):

n = 128: f = a^128 + a^7 + a^2 + a^1 + 1,

n = 256: f = a^256 + a^10 + a^5 + a^2 + 1.

The initial counter nonce ICN value for each message that is encrypted under the given initial key must be chosen in a unique manner.

6.3.4. CBC-ACPKM-Master Encryption Mode

This section defines a CBC-ACPKM-Master encryption mode that uses the ACPKM-Master internal re-keying mechanism for the periodical key transformation.

The CBC-ACPKM-Master encryption mode can be considered as the base encryption mode CBC (see [MODES]) extended by the ACPKM-Master re-keying mechanism.

The CBC-ACPKM-Master encryption mode can be used with the following parameters:

- 64 <= n <= 512.
- 128 <= k <= 512.
- The maximum message size m_max = N * (n * 2^{n/2-1} / k).

In the specification of the CBC-ACPKM-Master mode, the plaintext and ciphertext must be a sequence of one or more complete data blocks. If the data string to be encrypted does not initially satisfy this property, then it MUST be padded to form complete data blocks. The padding methods are out of the scope of this document. An example of a padding method can be found in Appendix A of [MODES].

The key material K[j] that is used for the j-th section processing is equal to K^j, |K^j| = k bits.

We use D_{K} to denote the decryption function that is a permutation inverse to E_{K}.

The CBC-ACPKM-Master mode encryption and decryption procedures are defined as follows:

```
+-------------------------------------------------------------------+
|  CBC-ACPKM-Master-Encrypt(N, K, T*, IV, P)                        |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initialization vector IV in V_n,                               |
|  - plaintext P = P_1 | ... | P_b, |P_b| = n, |P| <= m_max.      |
|  Output:                                                          |
|  - ciphertext C.                                                  |
|-------------------------------------------------------------------|
|  1. l = ceil(|P| / N)                                             |
|  2. K^1 | ... | K^l = ACPKM-Master(T*, K, k, l)                 |
|  3. C_0 = IV                                                      |
|  4. For j = 1, 2, ... , b do                                      |
|         i = ceil(j * n / N),                                      |
|         C_j = E_{K^i}(P_j (xor) C_{j-1})                          |
|  5. Return C = C_1 | ... | C_b                                    |
|-------------------------------------------------------------------+

+-------------------------------------------------------------------+
|  CBC-ACPKM-Master-Decrypt(N, K, T*, IV, C)                        |
|-------------------------------------------------------------------|
|  Input:                                                           |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - initialization vector IV in V_n,                               |
|  - ciphertext C = C_1 | ... | C_b, |C_b| = n, |C| <= m_max.     |
|  Output:                                                          |
|  - plaintext P.                                                   |
|-------------------------------------------------------------------|
|  1. l = ceil(|C| / N)                                             |
|  2. K^1 | ... | K^l = ACPKM-Master(T*, K, k, l)                 |
|  3. C_0 = IV                                                      |
|  4. For j = 1, 2, ... , b do                                      |
|         i = ceil(j * n / N)                                       |
|         P_j = D_{K^i}(C_j) (xor) C_{j-1}                          |
|  5. Return P = P_1 | ... | P_b                                    |
+-------------------------------------------------------------------+
```

The initialization vector IV for any particular execution of the encryption process must be unpredictable.

6.3.5. CFB-ACPKM-Master Encryption Mode

This section defines a CFB-ACPKM-Master encryption mode that uses the ACPKM-Master internal re-keying mechanism for the periodical key transformation.

The CFB-ACPKM-Master encryption mode can be considered as the base encryption mode CFB (see [MODES]) extended by the ACPKM-Master re-keying mechanism.

The CFB-ACPKM-Master encryption mode can be used with the following parameters:

- 64 <= n <= 512.
- 128 <= k <= 512.
- The maximum message size m_max = N * (n * 2^{n/2-1} / k).

The key material K[j] that is used for the j-th section processing is equal to K^j, |K^j| = k bits.

The CFB-ACPKM-Master mode encryption and decryption procedures are defined as follows:

```
+-------------------------------------------------------------+
|  CFB-ACPKM-Master-Encrypt(N, K, T*, IV, P)                  |
|-------------------------------------------------------------|
|  Input:                                                     |
|  - section size N,                                          |
|  - initial key K,                                           |
|  - master key frequency T*,                                 |
|  - initialization vector IV in V_n,                         |
|  - plaintext P = P_1 | ... | P_b, |P| <= m_max.            |
|  Output:                                                    |
|  - ciphertext C.                                            |
|-------------------------------------------------------------|
|  1. l = ceil(|P| / N)                                       |
|  2. K^1 | ... | K^l = ACPKM-Master(T*, K, k, l)           |
|  3. C_0 = IV                                                |
|  4. For j = 1, 2, ... , b - 1 do                            |
|         i = ceil(j * n / N),                                |
|         C_j = E_{K^i}(C_{j-1}) (xor) P_j                    |
|  5. C_b = MSB_{|P_b|}(E_{K^l}(C_{b-1})) (xor) P_b           |
|  6. Return C = C_1 | ... | C_b                              |
|-------------------------------------------------------------+
```

```
+-------------------------------------------------------------+
|  CFB-ACPKM-Master-Decrypt(N, K, T*, IV, C)                  |
|-------------------------------------------------------------|
|  Input:                                                     |
|  - section size N,                                          |
|  - initial key K,                                           |
|  - master key frequency T*,                                 |
|  - initialization vector IV in V_n,                         |
|  - ciphertext C = C_1 | ... | C_b, |C| <= m_max.           |
|  Output:                                                    |
|  - plaintext P.                                             |
|-------------------------------------------------------------|
|  1. l = ceil(|C| / N)                                       |
|  2. K^1 | ... | K^l = ACPKM-Master(T*, K, k, l)           |
|  3. C_0 = IV                                                |
|  4. For j = 1, 2, ... , b - 1 do                            |
|         i = ceil(j * n / N),                                |
|         P_j = E_{K^i}(C_{j-1}) (xor) C_j                    |
|  5. P_b = MSB_{|C_b|}(E_{K^l}(C_{b-1})) (xor) C_b           |
|  6. Return P = P_1 | ... | P_b                              |
+-------------------------------------------------------------+
```

The initialization vector IV for any particular execution of the encryption process must be unpredictable.

6.3.6. OMAC-ACPKM-Master Authentication Mode

This section defines an OMAC-ACPKM-Master message authentication code calculation mode that uses the ACPKM-Master internal re-keying mechanism for the periodical key transformation.

The OMAC-ACPKM-Master mode can be considered as the base message authentication code calculation mode OMAC1, which is also known as CMAC (see [RFC4493]), extended by the ACPKM-Master re-keying mechanism.

The OMAC-ACPKM-Master message authentication code calculation mode can be used with the following parameters:

- n in {64, 128, 256}.
- 128 <= k <= 512.
- The maximum message size m_max = N * (n * 2^{n/2-1} / (k + n)).

The key material K[j] that is used for one-section processing is equal to K^j | K^j_1, where |K^j| = k bits and |K^j_1| = n bits.

The following is a specification of the subkey generation process of OMAC:

```
+--------------------------------------------------------------------+
| Generate_Subkey(K1, r)                                             |
|--------------------------------------------------------------------|
| Input:                                                             |
|  - key K1.                                                         |
|  Output:                                                           |
|  - key SK.                                                         |
|--------------------------------------------------------------------|
|   1. If r = n, then return K1                                      |
|   2. If r < n, then                                                |
|          if MSB_1(K1) = 0                                          |
|              return K1 << 1                                        |
|          else                                                      |
|              return (K1 << 1) (xor) R_n                            |
+--------------------------------------------------------------------+
```

Here, R_n takes the following values:

- n = 64: R_{64} = 0^{59} | 11011.
- n = 128: R_{128} = 0^{120} | 10000111.
- n = 256: R_{256} = 0^{145} | 10000100101.

The OMAC-ACPKM-Master message authentication code calculation mode is defined as follows:

```
+-------------------------------------------------------------------+
| OMAC-ACPKM-Master(K, N, T*, M)                                    |
|-------------------------------------------------------------------|
| Input:                                                            |
|  - section size N,                                                |
|  - initial key K,                                                 |
|  - master key frequency T*,                                       |
|  - plaintext M = M_1 | ... | M_b, |M| <= m_max.                   |
|  Output:                                                          |
|  - message authentication code T.                                 |
|-------------------------------------------------------------------|
| 1. C_0 = 0^n                                                      |
| 2. l = ceil(|M| / N)                                              |
| 3. K^1 | K^1_1 | ... | K^l | K^l_1 =                            |
                  = ACPKM-Master(T*, K, (k + n), l)                 |
| 4. For j = 1, 2, ... , b - 1 do                                   |
|        i = ceil(j * n / N),                                       |
|        C_j = E_{K^i}(M_j (xor) C_{j-1})                           |
| 5. SK = Generate_Subkey(K^l_1, |M_b|)                            |
| 6. If |M_b| = n, then M*_b = M_b                                  |
|                  else M*_b = M_b | 1 | 0^{n - 1 -|M_b|}         |
| 7. T = E_{K^l}(M*_b (xor) C_{b-1} (xor) SK)                       |
| 8. Return T                                                       |
+-------------------------------------------------------------------+
```

7. Joint Usage of External and Internal Re-keying

Both external re-keying and internal re-keying have their own advantages and disadvantages, which are discussed in Section 1. For instance, using external re-keying can essentially limit the message length, while in the case of internal re-keying, the section size, which can be chosen as the maximal possible for operational properties, limits the number of separate messages. Therefore, the choice of re-keying mechanism (either external or internal) depends on particular protocol features. However, some protocols may have features that require the advantages of both the external and internal re-keying mechanisms: for example, the protocol mainly transmits short messages, but it must additionally support processing of very long messages. In such situations, it is necessary to use external and internal re-keying jointly, since these techniques negate each other’s disadvantages.

For composition of external and internal re-keying techniques, any mechanism described in Section 5 can be used with any mechanism described in Section 6.

For example, consider the GCM-ACPKM mode with external serial re-keying based on a KDF on a hash function. Denote the number of messages in each frame (in the case of the implicit approach to the key lifetime control) for external re-keying as a frame size.

Let L be a key lifetime limitation. The section size N for internal re-keying and the frame size q for external re-keying must be chosen in such a way that q * N must not exceed L.

Suppose that t messages (ICN_i, P_i, A_i), with initial counter nonce ICN_i, plaintext P_i, and additional authenticated data A_i will be processed before renegotiation.

For authenticated encryption of each message (ICN_i, P_i, A_i), i = 1, ..., t, the following algorithm can be applied:

1. j = ceil(i / q),
2. K^j = ExtSerialH(K, j),
3. C_i | T_i = GCM-ACPKM-Encrypt(N, K^j, ICN_i, P_i, A_i).

Note that nonces ICN_i that are used under the same frame key must be unique for each message.

8. Security Considerations

Re-keying should be used to increase a priori security properties of ciphers in hostile environments (e.g., with side-channel adversaries). If efficient attacks on a cipher are known, the cipher must not be used. Thus, re-keying cannot be used as a patch for vulnerable ciphers. Base cipher properties must be well analyzed because the security of re-keying mechanisms is based on the security of a block cipher as a pseudorandom function.

Re-keying is not intended to solve any postquantum security issues for symmetric cryptography, since the reduction of security caused by Grover’s algorithm is not connected with a size of plaintext transformed by a cipher -- only a negligible (sufficient for key uniqueness) material is needed -- and the aim of re-keying is to limit the size of plaintext transformed under one initial key.

Re-keying can provide backward security only if previous key material is securely deleted after usage by all parties.

9. IANA Considerations

This document has no IANA actions.

10. References

10.1. Normative References

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[DTLS] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[ESP] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, DOI 10.6028/NIST.SP.800-38D, November 2007, <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>.

[MODES] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", NIST Special Publication 800-38A, DOI 10.6028/NIST.SP.800-38A, December 2001.

[NISTSP800-108] National Institute of Standards and Technology, "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, October 2009, <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7836] Smyshlyaev, S., Ed., Alekseev, E., Oshkin, I., Popov, V., Leontiev, S., Podobaev, V., and D. Belyavsky, "Guidelines on the Cryptographic Algorithms to Accompany the Usage of Standards GOST R 34.10-2012 and GOST R 34.11-2012", RFC 7836, DOI 10.17487/RFC7836, March 2016, <https://www.rfc-editor.org/info/rfc7836>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SSH] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

10.2. Informative References

[AAOS2017] Ahmetzyanova, L., Alekseev, E., Oshkin, I., and S. Smyshlyaev, "Increasing the Lifetime of Symmetric Keys for the GCM Mode by Internal Re-keying", Cryptology ePrint Archive, Report 2017/697, 2017, <https://eprint.iacr.org/2017/697.pdf>.

[AbBell] Abdalla, M. and M. Bellare, "Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-keying Techniques", ASIACRYPT 2000, Lecture Notes in Computer Science, Volume 1976, pp. 546-559, DOI 10.1007/3-540-44448-3_42, October 2000.

[AESDUKPT] American National Standards Institute, "Retail Financial Services Symmetric Key Management - Part 3: Derived Unique Key Per Transaction", ANSI X9.24-3-2017, October 2017.

[FKK2005] Fu, K., Kamara, S., and T. Kohno, "Key Regression: Enabling Efficient Key Distribution for Secure Distributed Storage", November 2005, <https://homes.cs.washington.edu/~yoshi/papers/KR/NDSS06.pdf>.

[FPS2012] Faust, S., Pietrzak, K., and J. Schipper, "Practical Leakage-Resilient Symmetric Cryptography", Cryptographic Hardware and Embedded Systems (CHES), Lecture Notes in Computer Science, Volume 7428, pp. 213-232, DOI 10.1007/978-3-642-33027-8_13, 2012, <https://link.springer.com/content/pdf/10.1007%2F978-3-642-33027-8_13.pdf>.

[FRESHREKEYING] Dziembowski, S., Faust, S., Herold, G., Journault, A., Masny, D., and F. Standaert, "Towards Sound Fresh Re-Keying with Hard (Physical) Learning Problems", Cryptology ePrint Archive, Report 2016/573, June 2016, <https://eprint.iacr.org/2016/573>.

[GGM] Goldreich, O., Goldwasser, S., and S. Micali, "How to Construct Random Functions", Journal of the Association for Computing Machinery, Volume 33, No. 4, pp. 792-807, DOI 10.1145/6490.6503, October 1986, <https://dl.acm.org/citation.cfm?doid=6490.6503>.

[KMNT2003] Kim, Y., Maino, F., Narasimha, M., and G. Tsudik, "Secure Group Services for Storage Area Networks", IEEE Communications Magazine 41, Number 8, pp. 92-99, DOI 10.1109/SISW.2002.1183514, August 2003, <https://ieeexplore.ieee.org/document/1183514>.

[LDC] Heys, H., "A Tutorial on Linear and Differential Cryptanalysis", 2001, <https://citeseerx.ist.psu.edu/viewdoc/citations?doi=10.1.1.2.2759>.

[OWT] Joye, M. and S. Yen, "One-Way Cross-Trees and Their Applications", Public Key Cryptography (PKC), Lecture Notes in Computer Science, Volume 2274, DOI 10.1007/3-540-45664-3_25, February 2002, <https://link.springer.com/content/pdf/10.1007%2F3-540-45664-3_25.pdf>.

[P3] Alexander, P., "Subject: [Cfrg] Dynamic Key Changes on Encrypted Sessions. - Draft I-D Attached", message to the CFRG mailing list, 4 November 2017, <https://mailarchive.ietf.org/arch/msg/cfrg/ecTR3Hb-DFfrPCVmY0ghyYOEcxU>.

[Pietrzak2009]
Pietrzak, K., "A Leakage-Resilient Mode of Operation", EUROCRYPT 2009, Lecture Notes in Computer Science, Volume 5479, pp. 462-482, DOI 10.1007/978-3-642-01001-9_27, April 2009, <https://iacr.org/archive/eurocrypt2009/54790461/54790461.pdf>.

[SIGNAL] Perrin, T., Ed. and M. Marlinspike, "The Double Ratchet Algorithm", November 2016, <https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf>.

[Sweet32] Bhargavan, K. and G. Leurent, "On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 456-467, DOI 10.1145/2976749.2978423, October 2016, <https://sweet32.info/SWEET32_CCS16.pdf>.

[TAHA] Taha, M. and P. Schaumont, "Key Updating for Leakage Resiliency With Application to AES Modes of Operation", IEEE Transactions on Information Forensics and Security, DOI 10.1109/TIFS.2014.2383359, December 2014, <http://ieeexplore.ieee.org/document/6987331/>.

[TEMPEST] Ramsay, C. and J. Lohuis, "TEMPEST attacks against AES. Covertly stealing keys for 200 euro", June 2017, <https://www.fox-it.com/en/wp-content/uploads/sites/11/Tempest_attacks_against_AES.pdf>.

[U2F] Chang, D., Mishra, S., Sanadhya, S., and A. Singh, "On Making U2F Protocol Leakage-Resilient via Re-keying", Cryptology ePrint Archive, Report 2017/721, August 2017, <https://eprint.iacr.org/2017/721.pdf>.

Appendix A. Test Examples

A.1. Test Examples for External Re-keying

A.1.1. External Re-keying with a Parallel Construction

```
External re-keying with a parallel construction based on AES-256
****************************************************************
k = 256
t = 128

Initial key:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00

K^1:
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86
64 7D 5C D5 1C 3D 62 98 BC 09 B1 D8 64 EC D9 B1

K^2:
6F ED F5 D3 77 57 48 75 35 2B 5F 4D B6 5B E0 15
B8 02 92 32 D8 D3 8D 73 FE DC DD C6 C8 36 78 BD

K^3:
B6 40 24 85 A4 24 BD 35 B4 26 43 13 76 26 70 B6
5B F3 30 3D 3B 20 EB 14 D1 3B B7 91 74 E3 DB EC

...

K^126:
2F 3F 15 1B 53 88 23 CD 7D 03 FC 3D FD B3 57 5E
23 E4 1C 4E 46 FF 6B 33 34 12 27 84 EF 5D 82 23

K^127:
8E 51 31 FB 0B 64 BB D0 BC D4 C5 7B 1C 66 EF FD
97 43 75 10 6C AF 5D 5E 41 E0 17 F4 05 63 05 ED

K^128:
77 4F BF B3 22 60 C5 3B A3 8E FE B1 96 46 76 41
94 49 AF 84 2D 84 65 A7 F4 F7 2C DC A4 9D 84 F9

External re-keying with a parallel construction based on SHA-256
****************************************************************
k = 256
t = 128

label:
SHA2label
```

```
Initial key:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00

K^1:
C1 A1 4C A0 30 29 BE 43 9F 35 3C 79 1A 51 48 57
26 7A CD 5A E8 7D E7 D1 B2 E2 C7 AF A4 29 BD 35

K^2:
03 68 BB 74 41 2A 98 ED C4 7B 94 CC DF 9C F4 9E
A9 B8 A9 5F 0E DC 3C 1E 3B D2 59 4D D1 75 82 D4

K^3:
2F D3 68 D3 A7 8F 91 E6 3B 68 DC 2B 41 1D AC 80
0A C3 14 1D 80 26 3E 61 C9 0D 24 45 2A BD B1 AE

...

K^126:
55 AC 2B 25 00 78 3E D4 34 2B 65 0E 75 E5 8B 76
C8 04 E9 D3 B6 08 7D C0 70 2A 99 A4 B5 85 F1 A1

K^127:
77 4D 15 88 B0 40 90 E5 8C 6A D7 5D 0F CF 0A 4A
6C 23 F1 B3 91 B1 EF DF E5 77 64 CD 09 F5 BC AF

K^128:
E5 81 FF FB 0C 90 88 CD E5 F4 A5 57 B6 AB D2 2E
94 C3 42 06 41 AB C1 72 66 CC 2F 59 74 9C 86 B3
```

A.1.2. External Re-keying with a Serial Construction

```
External re-keying with a serial construction based on AES-256
**************************************************************
AES 256 examples:
k = 256
t = 128

Initial key:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00

K*_1:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00
```

```
K^1:
66 B8 BD E5 90 6C EC DF FA 8A B2 FD 92 84 EB F0
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86

K*_2:
64 7D 5C D5 1C 3D 62 98 BC 09 B1 D8 64 EC D9 B1
6F ED F5 D3 77 57 48 75 35 2B 5F 4D B6 5B E0 15

K^2:
66 B8 BD E5 90 6C EC DF FA 8A B2 FD 92 84 EB F0
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86

K*_3:
64 7D 5C D5 1C 3D 62 98 BC 09 B1 D8 64 EC D9 B1
6F ED F5 D3 77 57 48 75 35 2B 5F 4D B6 5B E0 15

K^3:
66 B8 BD E5 90 6C EC DF FA 8A B2 FD 92 84 EB F0
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86

...

K*_126:
64 7D 5C D5 1C 3D 62 98 BC 09 B1 D8 64 EC D9 B1
6F ED F5 D3 77 57 48 75 35 2B 5F 4D B6 5B E0 15

K^126:
66 B8 BD E5 90 6C EC DF FA 8A B2 FD 92 84 EB F0
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86

K*_127:
64 7D 5C D5 1C 3D 62 98 BC 09 B1 D8 64 EC D9 B1
6F ED F5 D3 77 57 48 75 35 2B 5F 4D B6 5B E0 15

K^127:
66 B8 BD E5 90 6C EC DF FA 8A B2 FD 92 84 EB F0
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86

K*_128:
64 7D 5C D5 1C 3D 62 98 BC 09 B1 D8 64 EC D9 B1
6F ED F5 D3 77 57 48 75 35 2B 5F 4D B6 5B E0 15

K^128:
66 B8 BD E5 90 6C EC DF FA 8A B2 FD 92 84 EB F0
51 16 8A B6 C8 A8 38 65 54 85 31 A5 D2 BA C3 86
```

External re-keying with a serial construction based on SHA-256
****

```
k = 256
t = 128

Initial key:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00

label1:
SHA2label1

label2:
SHA2label2

K*_1:
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0F 0E 0D 0C 0B 0A 09 08 07 06 05 04 03 02 01 00

K^1:
2D A8 D1 37 6C FD 52 7F F7 36 A4 E2 81 C6 0A 9B
F3 8E 66 97 ED 70 4F B5 FB 10 33 CC EC EE D5 EC

K*_2:
14 65 5A D1 7C 19 86 24 9B D3 56 DF CC BE 73 6F
52 62 4A 9D E3 CC 40 6D A9 48 DA 5C D0 68 8A 04

K^2:
2F EA 8D 57 2B EF B8 89 42 54 1B 8C 1B 3F 8D B1
84 F9 56 C7 FE 01 11 99 1D FB 98 15 FE 65 85 CF

K*_3:
18 F0 B5 2A D2 45 E1 93 69 53 40 55 43 70 95 8D
70 F0 20 8C DF B0 5D 67 CD 1B BF 96 37 D3 E3 EB

K^3:
53 C7 4E 79 AE BC D1 C8 24 04 BF F6 D7 B1 AC BF
F9 C0 0E FB A8 B9 48 29 87 37 E1 BA E7 8F F7 92

...

K*_126:
A3 6D BF 02 AA 0B 42 4A F2 C0 46 52 68 8B C7 E6
5E F1 62 C3 B3 2F DD EF E4 92 79 5D BB 45 0B CA

K^126:
6C 4B D6 22 DC 40 48 0F 29 C3 90 B8 E5 D7 A7 34
23 4D 34 65 2C CE 4A 76 2C FE 2A 42 C8 5B FE 9A
```

```
K*_127:
84 5F 49 3D B8 13 1D 39 36 2B BE D3 74 8F 80 A1
05 A7 07 37 BA 15 72 E0 73 49 C2 67 5D 0A 28 A1

K^127:
57 F0 BD 5A B8 2A F3 6B 87 33 CF F7 22 62 B4 D0
F0 EE EF E1 50 74 E5 BA 13 C1 23 68 87 36 29 A2

K*_128:
52 F2 0F 56 5C 9C 56 84 AF 69 AD 45 EE B8 DA 4E
7A A6 04 86 35 16 BA 98 E4 CB 46 D2 E8 9A C1 09

K^128:
9B DD 24 7D F3 25 4A 75 E0 22 68 25 68 DA 9D D5
C1 6D 2D 2B 4F 3F 1F 2B 5E 99 82 7F 15 A1 4F A4
```

A.2. Test Examples for Internal Re-keying

A.2.1. Internal Re-keying Mechanisms that Do Not Require a Master Key

```
CTR-ACPKM mode with AES-256
***************************
k = 256
n = 128
c = 64
N = 256

Initial key K:
00000:   88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
00010:   FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

Plaintext P:
00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88
00010:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A
00020:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
00030:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11
00040:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22
00050:   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33
00060:   55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33 44

ICN:
12 34 56 78 90 AB CE F0 A1 B2 C3 D4 E5 F0 01 12
23 34 45 56 67 78 89 90 12 13 14 15 16 17 18 19

D_1:
00000:   80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
```

```
   D_2:
   00000:   90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F

   Section_1

   Section key K^1:
   00000:   88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
   00010:   FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

   Input block CTR_1:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 00

   Output block G_1:
   00000:   FD 7E F8 9A D9 7E A4 B8 8D B8 B5 1C 1C 9D 6D D0

   Input block CTR_2:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 01

   Output block G_2:
   00000:   19 98 C5 71 76 37 FB 17 11 E4 48 F0 0C 0D 60 B2

   Section_2

   Section key K^2:
   00000:   F6 80 D1 21 2F A4 3D F4 EC 3A 91 DE 2A B1 6F 1B
   00010:   36 B0 48 8A 4F C1 2E 09 98 D2 E4 A8 88 E8 4F 3D

   Input block CTR_3:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 02

   Output block G_3:
   00000:   E4 88 89 4F B6 02 87 DB 77 5A 07 D9 2C 89 46 EA

   Input block CTR_4:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 03

   Output block G_4:
   00000:   BC 4F 87 23 DB F0 91 50 DD B4 06 C3 1D A9 7C A4

   Section_3

   Section key K^3:
   00000:   8E B9 7E 43 27 1A 42 F1 CA 8E E2 5F 5C C7 C8 3B
   00010:   1A CE 9E 5E D0 6A A5 3B 57 B9 6A CF 36 5D 24 B8

   Input block CTR_5:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 04
```

```
   Output block G_5:
   00000:   68 6F 22 7D 8F B2 9C BD 05 C8 C3 7D 22 FE 3B B7

   Input block CTR_6:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 05

   Output block G_6:
   00000:   C0 1B F9 7F 75 6E 12 2F 80 59 55 BD DE 2D 45 87

   Section_4

   Section key K^4:
   00000:   C5 71 6C C9 67 98 BC 2D 4A 17 87 B7 8A DF 94 AC
   00010:   E8 16 F8 0B DB BC AD 7D 60 78 12 9C 0C B4 02 F5

   Block number 7:

   Input block CTR_7:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 06

   Output block G_7:
   00000:   03 DE 34 74 AB 9B 65 8A 3B 54 1E F8 BD 2B F4 7D


   The result G = G_1 | G_2 | G_3 | G_4 | G_5 | G_6 | G_7:
   00000:   FD 7E F8 9A D9 7E A4 B8 8D B8 B5 1C 1C 9D 6D D0
   00010:   19 98 C5 71 76 37 FB 17 11 E4 48 F0 0C 0D 60 B2
   00020:   E4 88 89 4F B6 02 87 DB 77 5A 07 D9 2C 89 46 EA
   00030:   BC 4F 87 23 DB F0 91 50 DD B4 06 C3 1D A9 7C A4
   00040:   68 6F 22 7D 8F B2 9C BD 05 C8 C3 7D 22 FE 3B B7
   00050:   C0 1B F9 7F 75 6E 12 2F 80 59 55 BD DE 2D 45 87
   00060:   03 DE 34 74 AB 9B 65 8A 3B 54 1E F8 BD 2B F4 7D

   The result ciphertext C = P (xor) MSB_{|P|}(G):
   00000:   EC 5C CB DE 8C 18 D3 B8 72 56 68 D0 A7 37 F4 58
   00010:   19 89 E7 42 32 62 9D 60 99 7D E2 4B C0 E3 9F B8
   00020:   F5 AA BA 0B E3 64 F0 53 EE F0 BC 15 C2 76 4C EA
   00030:   9E 7C C3 76 BD 87 19 C9 77 0F CA 2D E2 A3 7C B5
   00040:   5B 2B 77 1B F8 3A 05 17 BE 04 2D 82 28 FE 2A 95
   00050:   84 4E 9F 08 FD F7 B8 94 4C B7 AA B7 DE 3C 67 B4
   00060:   56 B8 43 FC 32 31 DE 46 D5 AB 14 F8 AC 09 C7 39
```

```
GCM-ACPKM mode with AES-128
***************************
k = 128
n = 128
c = 32
N = 256

Initial key K:
00000:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Additional data A:
00000:   11 22 33

Plaintext:
00000:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00010:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00020:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ICN:
00000:   00 00 00 00 00 00 00 00 00 00 00 00

Number of sections: 2

Section key K^1:
00000:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Section key K^2:
00000:   15 1A 9F B0 B6 AC C5 97 6A FB 50 31 D1 DE C8 41

Encrypted GCTR_1 | GCTR_2 | GCTR_3:
00000:   03 88 DA CE 60 B6 A3 92 F3 28 C2 B9 71 B2 FE 78
00010:   F7 95 AA AB 49 4B 59 23 F7 FD 89 FF 94 8B C1 E0
00020:   D6 B3 12 46 E9 CE 9F F1 3A B3 42 7E E8 91 96 AD

Ciphertext C:
00000:   03 88 DA CE 60 B6 A3 92 F3 28 C2 B9 71 B2 FE 78
00010:   F7 95 AA AB 49 4B 59 23 F7 FD 89 FF 94 8B C1 E0
00020:   D6 B3 12 46 E9 CE 9F F1 3A B3 42 7E E8 91 96 AD

GHASH input:
00000:   11 22 33 00 00 00 00 00 00 00 00 00 00 00 00 00
00010:   03 88 DA CE 60 B6 A3 92 F3 28 C2 B9 71 B2 FE 78
00020:   F7 95 AA AB 49 4B 59 23 F7 FD 89 FF 94 8B C1 E0
00030:   D6 B3 12 46 E9 CE 9F F1 3A B3 42 7E E8 91 96 AD
00040:   00 00 00 00 00 00 00 18 00 00 00 00 00 00 01 80

GHASH output S:
00000:   E8 ED E9 94 9A DD 55 30 B0 F4 4E F5 00 FC 3E 3C
```

```
   Authentication tag  T:
   00000:   B0 0F 15 5A 60 A3 65 51 86 8B 53 A2 A4 1B 7B 66

   The result C | T:
   00000:   03 88 DA CE 60 B6 A3 92 F3 28 C2 B9 71 B2 FE 78
   00010:   F7 95 AA AB 49 4B 59 23 F7 FD 89 FF 94 8B C1 E0
   00020:   D6 B3 12 46 E9 CE 9F F1 3A B3 42 7E E8 91 96 AD
   00030:   B0 0F 15 5A 60 A3 65 51 86 8B 53 A2 A4 1B 7B 66
```

A.2.2. Internal Re-keying Mechanisms with a Master Key

```
   CTR-ACPKM-Master mode with AES-256
   **********************************
   k = 256
   n = 128
   c for CTR-ACPKM mode = 64
   c for CTR-ACPKM-Master mode = 64
   N = 256
   T* = 512

   Initial key K:
   00000:   88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
   00010:   FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

   Initial vector ICN:
   00000:   12 34 56 78 90 AB CE F0 A1 B2 C3 D4 E5 F0 01 12

   Plaintext P:
   00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88
   00010:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A
   00020:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
   00030:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11
   00040:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22
   00050:   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33
   00060:   55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33 44

   K^1 | K^2 | K^3 | K^4:
   00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
   00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60
   00020:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
   00030:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3
   00040:   E8 76 2B 30 8B 08 EB CE 3E 93 9A C2 C0 3E 76 D4
   00050:   60 9A AB D9 15 33 13 D3 CF D3 94 E7 75 DF 3A 94
   00060:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
   00070:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12
```

```
   Section_1

   K^1:
   00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
   00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60

   Input block CTR_1:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 00

   Output block G_1:
   00000:   8C A2 B6 82 A7 50 65 3F 8E BF 08 E7 9F 99 4D 5C

   Input block CTR_2:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 01

   Output block G_2:
   00000:   F6 A6 A5 BA 58 14 1E ED 23 DC 31 68 D2 35 89 A1


   Section_2

   K^2:
   00000:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
   00010:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3

   Input block CTR_3:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 02

   Output block G_3:
   00000:   4A 07 5F 86 05 87 72 94 1D 8E 7D F8 32 F4 23 71

   Input block CTR_4:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 03

   Output block G_4:
   00000:   23 35 66 AF 61 DD FE A7 B1 68 3F BA B0 52 4A D7


   Section_3

   K^3:
   00000:   E8 76 2B 30 8B 08 EB CE 3E 93 9A C2 C0 3E 76 D4
   00010:   60 9A AB D9 15 33 13 D3 CF D3 94 E7 75 DF 3A 94

   Input block CTR_5:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 04
```

```
   Output block G_5:
   00000:   A8 09 6D BC E8 BB 52 FC DE 6E 03 70 C1 66 95 E8

   Input block CTR_6:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 05

   Output block G_6:
   00000:   C6 E3 6E 8E 5B 82 AA C4 A6 6C 14 8D B1 F6 9B EF


   Section_4

   K^4:
   00000:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
   00010:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12

   Input block CTR_7:
   00000:   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 06

   Output block G_7:
   00000:   82 2B E9 07 96 37 44 95 75 36 3F A7 07 F8 40 22


   The result G = G_1 | G_2 | G_3 | G_4 | G_5 | G_6 | G_7:
   00000:   8C A2 B6 82 A7 50 65 3F 8E BF 08 E7 9F 99 4D 5C
   00010:   F6 A6 A5 BA 58 14 1E ED 23 DC 31 68 D2 35 89 A1
   00020:   4A 07 5F 86 05 87 72 94 1D 8E 7D F8 32 F4 23 71
   00030:   23 35 66 AF 61 DD FE A7 B1 68 3F BA B0 52 4A D7
   00040:   A8 09 6D BC E8 BB 52 FC DE 6E 03 70 C1 66 95 E8
   00050:   C6 E3 6E 8E 5B 82 AA C4 A6 6C 14 8D B1 F6 9B EF
   00060:   82 2B E9 07 96 37 44 95 75 36 3F A7 07 F8 40 22

   The result ciphertext C = P (xor) MSB_{|P|}(G):
   00000:   9D 80 85 C6 F2 36 12 3F 71 51 D5 2B 24 33 D4 D4
   00010:   F6 B7 87 89 1C 41 78 9A AB 45 9B D3 1E DB 76 AB
   00020:   5B 25 6C C2 50 E1 05 1C 84 24 C6 34 DC 0B 29 71
   00030:   01 06 22 FA 07 AA 76 3E 1B D3 F3 54 4F 58 4A C6
   00040:   9B 4D 38 DA 9F 33 CB 56 65 A2 ED 8F CB 66 84 CA
   00050:   82 B6 08 F9 D3 1B 00 7F 6A 82 EB 87 B1 E7 B9 DC
   00060:   D7 4D 9E 8F 0F 9D FF 59 9B C9 35 A7 16 DA 73 66
```

```
GCM-ACPKM-Master mode with AES-256
**********************************
k = 192
n = 128
c for the CTR-ACPKM mode = 64
c for the GCM-ACPKM-Master mode = 32
T* = 384
N = 256

Initial key K:
00000:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00010:   00 00 00 00 00 00 00 00

Additional data A:
00000:   11 22 33

Plaintext:
00000:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00010:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00020:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00030:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00040:   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ICN:
00000:   00 00 00 00 00 00 00 00 00 00 00 00

Number of sections: 3

K^1 | K^2 | K^3:
00000:   93 BA AF FB 35 FB E7 39 C1 7C 6A C2 2E EC F1 8F
00010:   7B 89 F0 BF 8B 18 07 05 96 48 68 9F 36 A7 65 CC
00020:   CD 5D AC E2 0D 47 D9 18 D7 86 D0 41 A8 3B AB 99
00030:   F5 F8 B1 06 D2 71 78 B1 B0 08 C9 99 0B 72 E2 87
00040:   5A 2D 3C BE F1 6E 67 3C

Encrypted GCTR_1 | ... | GCTR_5
00000:   43 FA 71 81 64 B1 E3 D7 1E 7B 65 39 A7 02 1D 52
00010:   69 9B 9E 1B 43 24 B7 52 95 74 E7 90 F2 BE 60 E8
00020:   11 62 C9 90 2A 2B 77 7F D9 6A D6 1A 99 E0 C6 DE
00030:   4B 91 D4 29 E3 1A 8C 11 AF F0 BC 47 F6 80 AF 14
00040:   40 1C C1 18 14 63 8E 76 24 83 37 75 16 34 70 08

Ciphertext C:
00000:   43 FA 71 81 64 B1 E3 D7 1E 7B 65 39 A7 02 1D 52
00010:   69 9B 9E 1B 43 24 B7 52 95 74 E7 90 F2 BE 60 E8
00020:   11 62 C9 90 2A 2B 77 7F D9 6A D6 1A 99 E0 C6 DE
00030:   4B 91 D4 29 E3 1A 8C 11 AF F0 BC 47 F6 80 AF 14
00040:   40 1C C1 18 14 63 8E 76 24 83 37 75 16 34 70 08
```

```
GHASH input:
00000:   11 22 33 00 00 00 00 00 00 00 00 00 00 00 00 00
00010:   43 FA 71 81 64 B1 E3 D7 1E 7B 65 39 A7 02 1D 52
00020:   69 9B 9E 1B 43 24 B7 52 95 74 E7 90 F2 BE 60 E8
00030:   11 62 C9 90 2A 2B 77 7F D9 6A D6 1A 99 E0 C6 DE
00040:   4B 91 D4 29 E3 1A 8C 11 AF F0 BC 47 F6 80 AF 14
00050:   40 1C C1 18 14 63 8E 76 24 83 37 75 16 34 70 08
00060:   00 00 00 00 00 00 00 18 00 00 00 00 00 00 02 80

GHASH output S:
00000:   6E A3 4B D5 6A C5 40 B7 3E 55 D5 86 D1 CC 09 7D

Authentication tag  T:
00050:   CC 3A BA 11 8C E7 85 FD 77 78 94 D4 B5 20 69 F8

The result C | T:
00000:   43 FA 71 81 64 B1 E3 D7 1E 7B 65 39 A7 02 1D 52
00010:   69 9B 9E 1B 43 24 B7 52 95 74 E7 90 F2 BE 60 E8
00020:   11 62 C9 90 2A 2B 77 7F D9 6A D6 1A 99 E0 C6 DE
00030:   4B 91 D4 29 E3 1A 8C 11 AF F0 BC 47 F6 80 AF 14
00040:   40 1C C1 18 14 63 8E 76 24 83 37 75 16 34 70 08
00050:   CC 3A BA 11 8C E7 85 FD 77 78 94 D4 B5 20 69 F8


CBC-ACPKM-Master mode with AES-256
**********************************
k = 256
n = 128
c for the CTR-ACPKM mode = 64
N = 256
T* = 512

Initial key K:
00000:   88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
00010:   FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

Initial vector IV:
00000:   12 34 56 78 90 AB CE F0 A1 B2 C3 D4 E5 F0 01 12

Plaintext P:
00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88
00010:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A
00020:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
00030:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11
00040:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22
00050:   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33
00060:   55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33 44
```

```
K^1 | K^2 | K^3 | K^4:
00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60
00020:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
00030:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3
00040:   E8 76 2B 30 8B 08 EB CE 3E 93 9A C2 C0 3E 76 D4
00050:   60 9A AB D9 15 33 13 D3 CF D3 94 E7 75 DF 3A 94
00060:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
00070:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12

Section_1

K^1:
00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60

Plaintext block P_1:
00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88

Input block P_1 (xor) C_0:
00000:   03 16 65 3C C5 CD B9 F0 5E 5C 1E 18 5E 5A 98 9A

Output block C_1:
00000:   59 CB 5B CA C2 69 2C 60 0D 46 03 A0 C7 40 C9 7C

Plaintext block P_2:
00000:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A

Input block P_2 (xor) C_1:
00000:   59 DA 79 F9 86 3C 4A 17 85 DF A9 1B 0B AE 36 76

Output block C_2:
00000:   80 B6 02 74 54 8B F7 C9 78 1F A1 05 8B F6 8B 42

Section_2

K^2:
00000:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
00010:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3

Plaintext block P_3:
00000:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00

Input block P_3 (xor) C_2:
00000:   91 94 31 30 01 ED 80 41 E1 B5 1A C9 65 09 81 42

Output block C_3:
00000:   8C 24 FB CF 68 15 B1 AF 65 FE 47 75 95 B4 97 59
```

```
   Plaintext block P_4:
   00000:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11

   Input block P_4 (xor) C_3:
   00000:   AE 17 BF 9A 0E 62 39 36 CF 45 8B 9B 6A BE 97 48

   Output block C_4:
   00000:   19 65 A5 00 58 0D 50 23 72 1B E9 90 E1 83 30 E9

   Section_3

   K^3:
   00000:   E8 76 2B 30 8B 08 EB CE 3E 93 9A C2 C0 3E 76 D4
   00010:   60 9A AB D9 15 33 13 D3 CF D3 94 E7 75 DF 3A 94

   Plaintext block P_5:
   00000:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22

   Input block P_5 (xor) C_4:
   00000:   2A 21 F0 66 2F 85 C9 89 C9 D7 07 6F EB 83 21 CB

   Output block C_5:
   00000:   56 D8 34 F4 6F 0F 4D E6 20 53 A9 5C B5 F6 3C 14

   Plaintext block P_6:
   00000:   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33

   Input block P_6 (xor) C_5:
   00000:   12 8D 52 83 E7 96 E7 5D EC BD 56 56 B5 E7 1E 27

   Output block C_6:
   00000:   66 68 2B 8B DD 6E B2 7E DE C7 51 D6 2F 45 A5 45

   Section_4

   K^4:
   00000:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
   00010:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12

   Plaintext block P_7:
   00000:   55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33 44

   Input block P_7 (xor) C_6:
   00000:   33 0E 5C 03 44 C4 09 B2 30 38 5B D6 3E 67 96 01

   Output block C_7:
   00000:   7F 4D 87 F9 CA E9 56 09 79 C4 FA FE 34 0B 45 34
```

```
Ciphertext C:
00000:   59 CB 5B CA C2 69 2C 60 0D 46 03 A0 C7 40 C9 7C
00010:   80 B6 02 74 54 8B F7 C9 78 1F A1 05 8B F6 8B 42
00020:   8C 24 FB CF 68 15 B1 AF 65 FE 47 75 95 B4 97 59
00030:   19 65 A5 00 58 0D 50 23 72 1B E9 90 E1 83 30 E9
00040:   56 D8 34 F4 6F 0F 4D E6 20 53 A9 5C B5 F6 3C 14
00050:   66 68 2B 8B DD 6E B2 7E DE C7 51 D6 2F 45 A5 45
00060:   7F 4D 87 F9 CA E9 56 09 79 C4 FA FE 34 0B 45 34


CFB-ACPKM-Master mode with AES-256
**********************************
k = 256
n = 128
c for the CTR-ACPKM mode = 64
N = 256
T* = 512

Initial key K:
00000:   88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
00010:   FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

Initial vector IV:
00000:   12 34 56 78 90 AB CE F0 A1 B2 C3 D4 E5 F0 01 12

Plaintext P:
00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88
00010:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A
00020:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
00030:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11
00040:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22
00050:   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33
00060:   55 66 77 88 99 AA BB CC

K^1 | K^2 | K^3 | K^4
00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60
00020:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
00030:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3
00040:   E8 76 2B 30 8B 08 EB CE 3E 93 9A C2 C0 3E 76 D4
00050:   60 9A AB D9 15 33 13 D3 CF D3 94 E7 75 DF 3A 94
00060:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
00070:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12
```

```
   Section_1

   K^1:
   00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
   00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60

   Plaintext block P_1:
   00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88

   Encrypted block E_{K^1}(C_0):
   00000:   1C 39 9D 59 F8 5D 91 91 A9 D2 12 9F 63 15 90 03

   Output block C_1 = E_{K^1}(C_0) (xor) P_1:
   00000:   0D 1B AE 1D AD 3B E6 91 56 3C CF 53 D8 BF 09 8B

   Plaintext block P_2:
   00000:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A

   Encrypted block E_{K^1}(C_1):
   00000:   6B A2 C5 42 52 69 C6 0B 15 14 06 87 90 46 F6 2E

   Output block C_2 = E_{K^1}(C_1) (xor) P_2:
   00000:   6B B3 E7 71 16 3C A0 7C 9D 8D AC 3C 5C A8 09 24

   Section_2

   K^2:
   00000:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
   00010:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3

   Plaintext block P_3:
   00000:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00

   Encrypted block E_{K^2}(C_2):
   00000:   95 45 5F DB C3 9E 0A 13 9F CB 10 F5 BD 79 A3 88

   Output block C_3 = E_{K^2}(C_2) (xor) P_3:
   00000:   84 67 6C 9F 96 F8 7D 9B 06 61 AB 39 53 86 A9 88

   Plaintext block P_4:
   00000:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11

   Encrypted block E_{K^2}(C_3):
   00000:   E0 AA 32 5D 80 A4 47 95 BA 42 BF 63 F8 4A C8 B2

   Output block C_4 = E_{K^2}(C_3) (xor) P_4:
   00000:   C2 99 76 08 E6 D3 CF 0C 10 F9 73 8D 07 40 C8 A3
```

```
Section_3

K^3:
00000:   E8 76 2B 30 8B 08 EB CE 3E 93 9A C2 C0 3E 76 D4
00010:   60 9A AB D9 15 33 13 D3 CF D3 94 E7 75 DF 3A 94

Plaintext block P_5:
00000:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22

Encrypted block E_{K^3}(C_4):
00000:   FE 42 8C 70 C2 51 CE 13 36 C1 BF 44 F8 49 66 89

Output block C_5 = E_{K^3}(C_4) (xor) P_5:
00000:   CD 06 D9 16 B5 D9 57 B9 8D 0D 51 BB F2 49 77 AB

Plaintext block P_6:
00000:   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33

Encrypted block E_{K^3}(C_5):
00000:   01 24 80 87 86 18 A5 43 11 0A CC B5 0A E5 02 A3

Output block C_6 = E_{K^3}(C_5) (xor) P_6:
00000:   45 71 E6 F0 0E 81 0F F8 DD E4 33 BF 0A F4 20 90

Section_4

K^4:
00000:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
00010:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12

Plaintext block P_7:
00000:   55 66 77 88 99 AA BB CC

Encrypted block MSB_{|P_7|}(E_{K^4}(C_6)):
00000:   97 5C 96 37 55 1E 8C 7F

Output block C_7 = MSB_{|P_7|}(E_{K^4}(C_6)) (xor) P_7
00000:   C2 3A E1 BF CC B4 37 B3

Ciphertext C:
00000:   0D 1B AE 1D AD 3B E6 91 56 3C CF 53 D8 BF 09 8B
00010:   6B B3 E7 71 16 3C A0 7C 9D 8D AC 3C 5C A8 09 24
00020:   84 67 6C 9F 96 F8 7D 9B 06 61 AB 39 53 86 A9 88
00030:   C2 99 76 08 E6 D3 CF 0C 10 F9 73 8D 07 40 C8 A3
00040:   CD 06 D9 16 B5 D9 57 B9 8D 0D 51 BB F2 49 77 AB
00050:   45 71 E6 F0 0E 81 0F F8 DD E4 33 BF 0A F4 20 90
00060:   C2 3A E1 BF CC B4 37 B3
```

```
   OMAC-ACPKM-Master mode with AES-256
   ***********************************
   k = 256
   n = 128
   c for the CTR-ACPKM mode = 64
   N = 256
   T* = 768

   Initial key K:
   00000:   88 99 AA BB CC DD EE FF 00 11 22 33 44 55 66 77
   00010:   FE DC BA 98 76 54 32 10 01 23 45 67 89 AB CD EF

   Plaintext M:
   00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88
   00010:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A
   00020:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
   00030:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11
   00040:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22

   K^1 | K^1_1 | K^2 | K^2_1 | K^3 | K^3_1:
   00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
   00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60
   00020:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0
   00030:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3
   00040:   9D CC 66 42 0D FF 45 5B 21 F3 93 F0 D4 D6 6E 67
   00050:   BB 1B 06 0B 87 66 6D 08 7A 9D A7 49 55 C3 5B 48
   00060:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
   00070:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12
   00080:   78 21 C7 C7 6C BD 79 63 56 AC F8 8E 69 6A 00 07

   Section_1

   K^1:
   00000:   9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
   00010:   39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60

   K^1_1:
   00000:   77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0

   Plaintext block M_1:
   00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88

   Input block M_1 (xor) C_0:
   00000:   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88

   Output block C_1:
   00000:   0B A5 89 BF 55 C1 15 42 53 08 89 76 A0 FE 24 3E
```

```
   Plaintext block M_2:
   00000:   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A

   Input block M_2 (xor) C_1:
   00000:   0B B4 AB 8C 11 94 73 35 DB 91 23 CD 6C 10 DB 34

   Output block C_2:
   00000:   1C 53 DD A3 6D DC E1 17 ED 1F 14 09 D8 6A F3 2C

   Section_2

   K^2:
   00000:   AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3
   00010:   9D CC 66 42 0D FF 45 5B 21 F3 93 F0 D4 D6 6E 67

   K^2_1:
   00000:   BB 1B 06 0B 87 66 6D 08 7A 9D A7 49 55 C3 5B 48

   Plaintext block M_3:
   00000:   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00

   Input block M_3 (xor) C_2:
   00000:   0D 71 EE E7 38 BA 96 9F 74 B5 AF C5 36 95 F9 2C

   Output block C_3:
   00000:   4E D4 BC A6 CE 6D 6D 16 F8 63 85 13 E0 48 59 75

   Plaintext block M_4:
   00000:   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11

   Input block M_4 (xor) C_3:
   00000:   6C E7 F8 F3 A8 1A E5 8F 52 D8 49 FD 1F 42 59 64

   Output block C_4:
   00000:   B6 83 E3 96 FD 30 CD 46 79 C1 8B 24 03 82 1D 81

   Section_3

   K^3:
   00000:   F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
   00010:   2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12

   K^3_1:
   00000:   78 21 C7 C7 6C BD 79 63 56 AC F8 8E 69 6A 00 07

   MSB1(K1) == 0 -> K2 = K1 << 1
```

```
K1:
00000:   78 21 C7 C7 6C BD 79 63 56 AC F8 8E 69 6A 00 07

K2:
00000:   F0 43 8F 8E D9 7A F2 C6 AD 59 F1 1C D2 D4 00 0E

Plaintext M_5:
00000:   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22

Using K1, padding is not required

Input block M_5 (xor) C_4:
00000:   FD E6 71 37 E6 05 2D 8F 94 A1 9D 55 60 E8 0C A4

Output block C_5:
00000:   B3 AD B8 92 18 32 05 4C 09 21 E7 B8 08 CF A0 B8

Message authentication code T:
00000:   B3 AD B8 92 18 32 05 4C 09 21 E7 B8 08 CF A0 B8
```

Acknowledgments

We thank Mihir Bellare, Scott Fluhrer, Dorothy Cooley, Yoav Nir, Jim Schaad, Paul Hoffman, Dmitry Belyavsky, Yaron Sheffer, Alexey Melnikov, and Spencer Dawkins for their useful comments.

Contributors

Russ Housley
Vigil Security, LLC
housley@vigilsec.com

Evgeny Alekseev
CryptoPro
alekseev@cryptopro.ru

Ekaterina Smyshlyaeva
CryptoPro
ess@cryptopro.ru

Shay Gueron
University of Haifa, Israel
Intel Corporation, Israel Development Center, Israel
shay.gueron@gmail.com

Daniel Fox Franke
Akamai Technologies
dfoxfranke@gmail.com

Lilia Ahmetzyanova
CryptoPro
lah@cryptopro.ru

Author’s Address

Stanislav Smyshlyaev (editor)
CryptoPro
18, Suschevskiy val
Moscow 127018
Russian Federation

Phone: +7 (495) 995-48-20
Email: svs@cryptopro.ru