                    Re-keying Mechanisms for Symmetric Keys
                         draft-irtf-cfrg-re-keying-08

Abstract

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

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

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

   This Internet-Draft will expire on April 12, 2018.

Table of Contents

1. Introduction

A certain maximum amount of data can be safely encrypted when
encryption is performed under a single key. This amount is called
"key lifetime" and can be calculated from the following
considerations:

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 modes restrictions derived from such methods are of
        order $2^{n/2}$. [Sweet32] is an example of 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 block cipher
        features. Limitations resulting from these considerations are
        usually the most restrictive ones. [TEMPEST] is an example of
        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 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 as soon as the total
size of the processed plaintext under that key reaches 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 which is enclosed by area L (see
Figure 1).

```
            +-----------------------+
            |                   L |
            | +--------m---------+   |
            | |=================|    |
            | |=================|    |
            | q=================|    |        m * q <= L
            | |=================|    |
            | |=================|    |
            | +-----------------+    |
            +-----------------------+
```
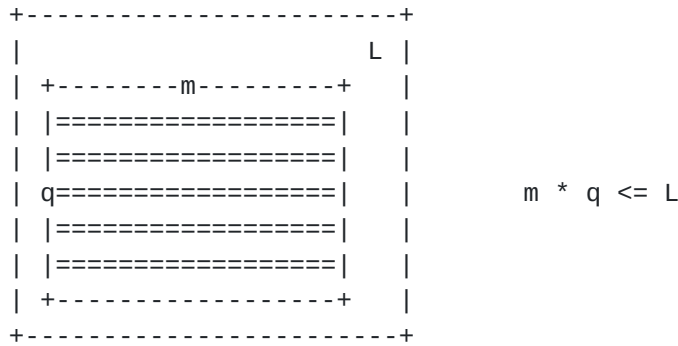
Figure 1: Graphic display of the key lifetime limitation

In practice, such amount of data that corresponds to limitation L may
not be enough. The simplest and obvious way 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 random number
generator and many other additional calculations, including resource-
intensive public key cryptography.

This specification presents two approaches to extend the lifetime of
a key while avoiding renegotiation: external and internal 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.

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 providing the possibility to limit the leakages (via side
channels) and by improving 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., that require lightweight cryptography, where
ciphers have a small block size, that imposes strict combinatorial
limitations). Moreover, mechanisms that use external and internal

re-keying may provide some properties of forward security and
potentially some protection against future attacks (by limiting the
number of plaintext-ciphertext pairs that an adversary can collect).

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

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;
        substrings and string components are enumerated from right to
        left starting from one;

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

$Z_{\{2^n\}}$ ring of residues modulo 2^n;

$Int_s: V_s \rightarrow Z_{\{2^s\}}$ the transformation that maps a string a =
        $(a_s, \ldots , a_1)$, a in V_s, into the integer $Int_s(a) =$
        $2^{\{s-1\}} * a_s + \ldots + 2 * a_2 + a_1$;

$Vec_s: Z_{\{2^s\}} \rightarrow V_s$ the transformation inverse to the mapping
        Int_s;

$MSB_i: V_s \rightarrow V_i$ the transformation that maps the string a = $(a_s,$
        $\ldots , a_1)$ in V_s, into the string $MSB_i(a) = (a_s, \ldots ,$
        $a_{\{s-i+1\}})$ in V_i;

$LSB_i: V_s \rightarrow V_i$ the transformation that maps the string a = $(a_s,$
        $\ldots , a_1)$ in V_s, into the string $LSB_i(a) = (a_i, \ldots ,$
        $a_1)$ in V_i;

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;

a^s   denotes 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 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 block P_b, C_b may
be an incomplete block, 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. 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 for 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 loss in 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 negligible affect on the 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 case of restrictive key lifetime limitations the
   message sizes can become inconvenient due to impossibility of
   processing sufficiently large messages, so it could be necessary to
   perform additional fragmentation at the protocol level. E.g. 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 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 does not
       affect short messages transformation at all;

   4. it is transparent (works like any mode of operation): does not
       require changes of IV's and restarting 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. This vulnerability can be eliminated by
using different primitives for data processing and re-keying,
however, in this case the security of the whole scheme cannot be
reduced to standard notions like PRF or PRP, so security estimations
become more difficult and unclear.

Summing up the above-mentioned issues briefly:

1. If a protocol assumes processing long records (e.g., [CMS]),
    internal re-keying should be used. If a protocol assumes
    processing a significant amount 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 which allow out-of-order delivery and lost records
    (e.g., [DTLS], [ESP]) external re-keying should be used. If at
    the same time records are long enough, internal re-keying should
    be additionally used during each separate message processing.

For external re-keying:

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

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

3. In case of restrictive key lifetime limitations 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 increase the initial key
   lifetime by using a transformation of a data processing key (frame
   key) after processing a limited number of entire messages (frame).
   It 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), q * m <= L. As L1 limitation
   is usually much stronger than 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.

```
            <--------m------->
            +----------------+ ^ ^
            |================| | |
            |================| | |
    K-->|================| q1|
            |================| | |
            |============L1| | |
            +----------------+ v |
            |                |   |
            |                |   |
            |                |   q2
            |                |   |
            |                |   |
            |                |   |
            |                |   |
            |                |   |
            |                |   |
            |                |   |
            |          L2|   |
            +----------------+   v
```

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 goes
   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 in case
   of using external re-keying can be increased up to L2'.

```
            <--------m------->
       K      +----------------+
       |      |================|
       v      |================|
     K^1-->  |================|
       |      |================|
       |      |============L1|
       |      +----------------+
       |      |================|
       v      |================|
     K^2-->  |================|
       |      |================|
       |      |============L1|
       |      +----------------+
       |      |================|
       v      |================|
      ...     |      . . .     |
              |                |
              |                |
              |           L2|
            +----------------+
              |                |
      ...                 ...
              |           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 then 1 TB. Thus if an external re-keying
   mechanism is used, more then 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
   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 section key. For i in {1, 2, ... , t} the frame key K^i (see Figure 4 and Figure 5) should be transformed after processing q_i messages, where q_i can be calculated in accordance with one of the following two approaches:

o Explicit approach:
   q_i is such that $|M^{i,1}| + \ldots + |M^{i,q_i}| \le L$, $|M^{i,1}| + \ldots + |M^{i,q_i+1}| > L$.
   This approach allows to use the frame key K^i in almost optimal way but it can be applied only in case when messages cannot be lost or reordered (e.g., TLS records).

o 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 in case when messages may be lost or reordered (e.g., DTLS records).

5.2. Parallel Constructions

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

Maximum message size = m_max.

_____

```
                                m_max
                        <---------------->
                M^{1,1}   |===              |
                M^{1,2}   |===============  |
      +->K^1-->   ...              ...
      |         M^{1,q_1} |========         |
      |
      |
      |         M^{2,1}   |===============  |
      |         M^{2,2}   |=====            |
K-----|->K^2-->   ...              ...
      |         M^{2,q_2} |=========        |
      |
     ...
      |         M^{t,1}   |============     |
      |         M^{t,2}   |=============    |
      +->K^t-->   ...              ...
                M^{t,q_t} |==========       |
```

_____

Figure 4: External parallel re-keying mechanisms

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

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

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 HKDF

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 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 6.

```
                              K_root = K
                      _____|_____
                      |           ...          |
                      V                         V
                   K{1,1}                    K{1,W1}
                 _____|_____            _____|_____
                 |    ...     |            |    ...     |
                 V            V            V            V
              K{2,1}       K{2,W2}   K{2,(W1-1)*W2+1} K{2,W1*W2}
              __|__         __|__        __|__          __|__
              | ... |       | ... |      | ... |        | ... |
              V     V       V     V      V     V        V     V
            K{3,1}   ...      ...    ...  ...   ...      ...  K{3,W1*W2*W3}

      ...                                                  ...
      __|__                        ...                     __|__
     | ... |                                              | ... |
     V     V                                              V     V
  K{h,1}  K{h,Wh}                 K{h,(W1*...*W{h-1}-1)*Wh+1}  K{h,W1*...*Wh}
   //      \\                                      //       \\
  K^1       K^{Wh}            K^{(W1*...*W{h-1}-1)*Wh+1}      K^{W1*...*Wh}
  _____
```
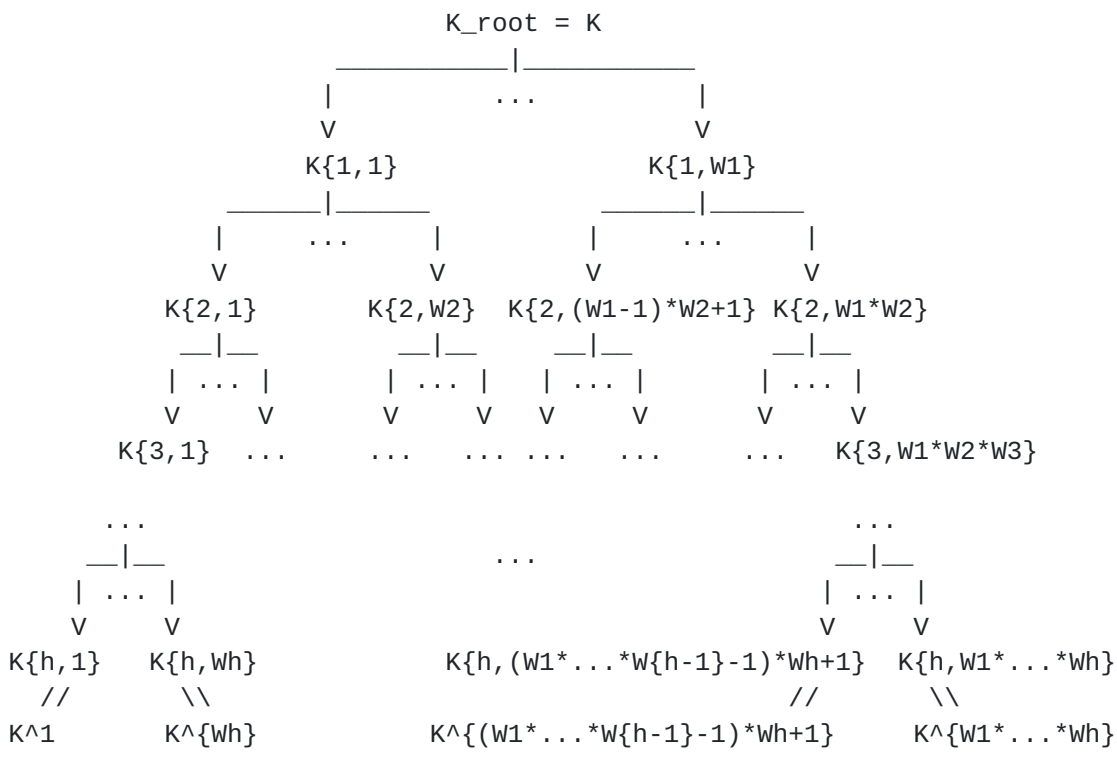
Figure 6: External Tree-based Mechanism

The height of tree h and the number of keys Wj, j in {1, ... , h},
which can be partitioned from "parent" key, are defined in accordance
with a specific protocol and key lifetime limitations for a 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 j-th level derivation
function that can be based on the block cipher function or on the
hash function and that 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 a j-th level derivation function that takes two
arguments (the parent key value and the integer in 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 amount of
messages (see Section 5.1).

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

Consider an example. Suppose h = 3, W1 = W2 = W3 = W and KDF_1,
KDF_2, KDF_3 are key derivation functions based on
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}.

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

5.3. Serial Constructions

The main idea behind external re-keying with a serial construction is
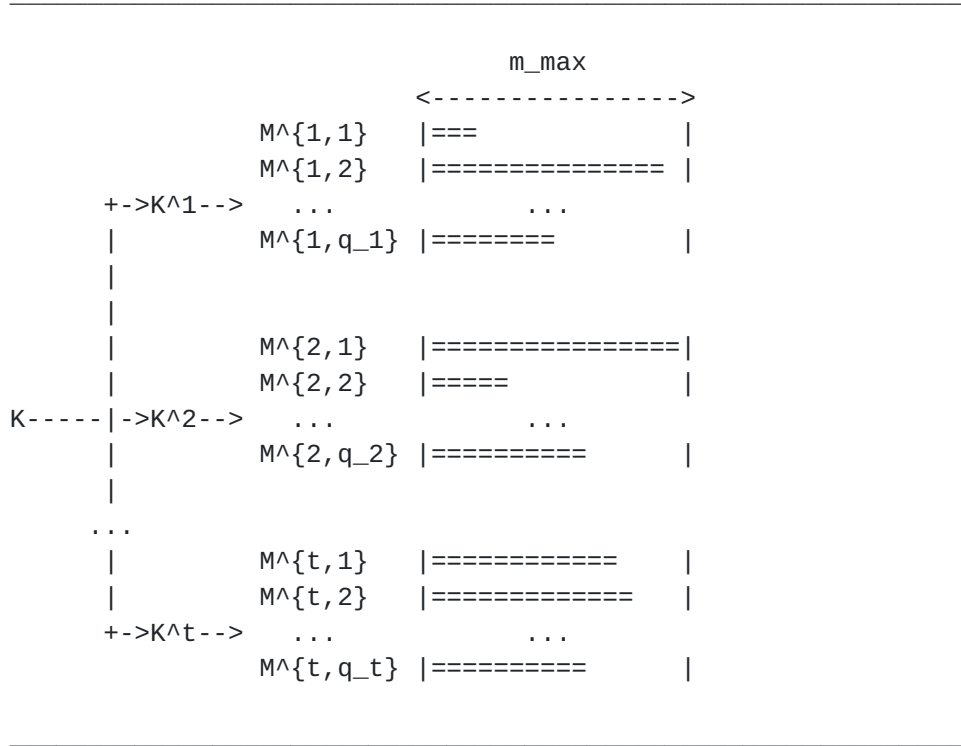presented in Figure 5:

Maximum message size = m_max.

```
                                          m_max
                                 <---------------->
                       M^{1,1}   |===             |
                       M^{1,2}   |============== |
 K*_1 = K --->K^1-->      ...              ...
    |                  M^{1,q_1} |========       |
    |
    |
    |                  M^{2,1}   |==============|
    v                  M^{2,2}   |=====          |
 K*_2 ------->K^2-->      ...              ...
    |                  M^{2,q_2} |==========     |
    |
   ...
    |                  M^{t,1}   |===========    |
    v                  M^{t,2}   |============   |
 K*_t ------->K^t-->      ...              ...
                       M^{t,q_t} |==========     |
```
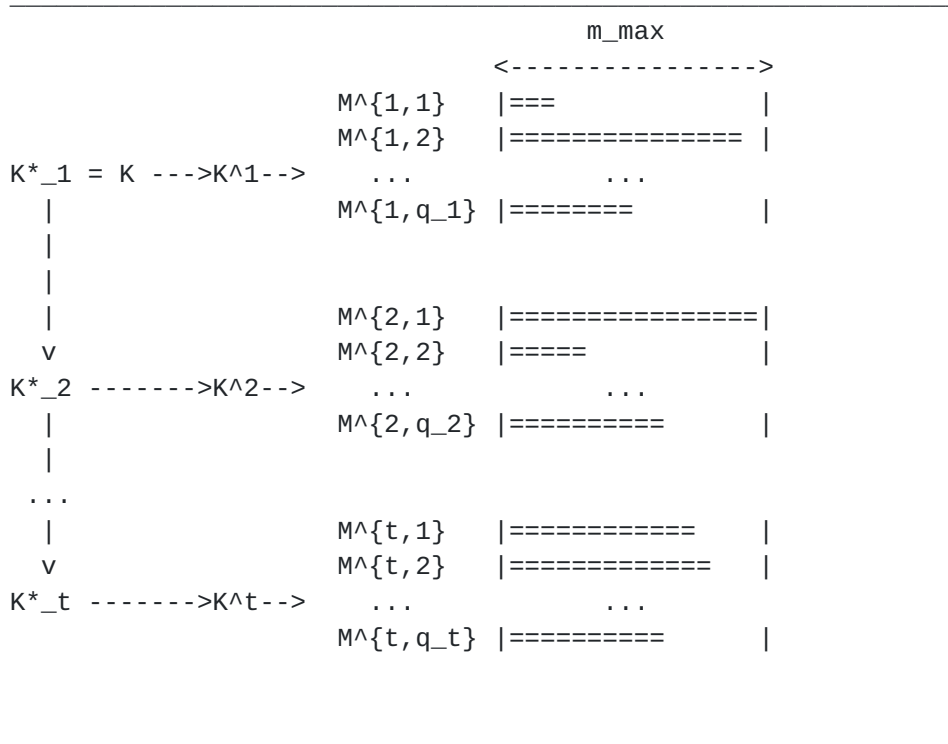
Figure 5: External serial re-keying mechanisms

The frame key K^i, i = 1, ... , t - 1, is updated after processing a certain amount 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 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 $J = \text{ceil}(k / n)$, 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 HKDF

   The frame key K^i is calculated using 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, described in [RFC5869], 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, TLS 1.3 updating
   traffic keys algorithm [TLSDraft]).

6. Internal Re-keying Mechanisms

   This section presents an approach to increase 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 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 amount 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, by analogy with the [Section 5](#) without re-keying the final key lifetime restriction, as displayed in Figure 7, is equal to L1 and only q messages of the length m1 can be safely processed.

```
            K
            |
            v
  ^ +---------------+-----------------------------------+
  | |=============L1|                              L2|
  | |===============|                                |
  q |===============|                                |
  | |===============|                                |
  | |===============|                                |
  v +---------------+-----------------------------------+
    <-------m1------->
    <----------------------------m2---------------------->
```
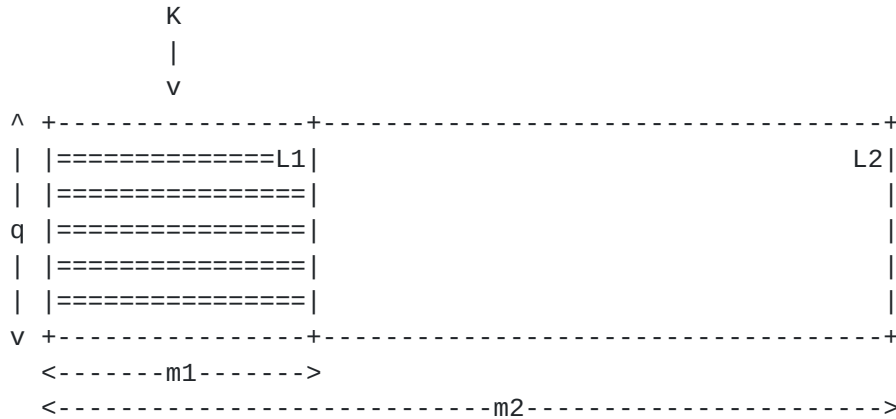
Figure 7: Basic principles of message processing without internal re-keying

Suppose that the safety margin for the protocol P is fixed and 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 base mode of operation without re-keying (thus, L2 < L2'). Hence, as displayed in Figure 8, the resulting key lifetime limitation in case of using internal re-keying can be increased up to L2'.

```
  K-----> K^1-------------> K^2 ----------->  . . .
          |                 |
          v                 v
^ +---------------+---------------+------------------+--...--+
| |==============L1|==============L1|======            L2|   L2'|
| |===============|===============|======            |      |
q |===============|===============|======  . . .      |      |
| |===============|===============|======            |      |
| |===============|===============|======            |      |
v +---------------+---------------+------------------+--...--+
   <-------N-------->
```
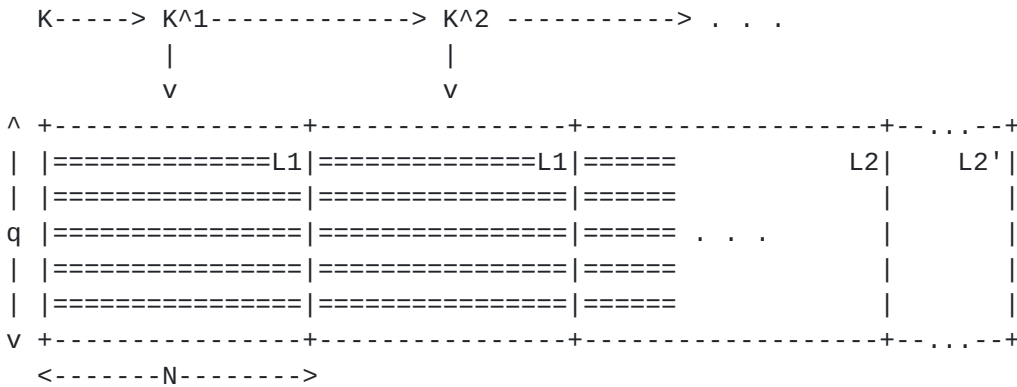
Figure 8: Basic principles of message processing with internal re-keying

   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 parameter N should be selected for a
   particular protocol as maximum possible to provide necessary key
   lifetime for the security models that are considered.

   Consider an example. Suppose L1 = 128 MB and L2 = 10 TB. Let the
   message size in the protocol be large/unlimited (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 4
   messages in case 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 then 10 TB (in the case 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, N is a section size (fixed parameter). Suppose M^{i}_1
   is the first section of message M^{i}, i = 1, ... , q (see Figure 9
   and Figure 10), then the parameter q can be calculated in accordance
   with one of the following two approaches:

   o 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 to use the section key K^i in an almost
      optimal way but it can be applied only in case when messages
      cannot be lost or reordered (e.g., TLS records).

   o 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 or greater than section size N and so it can be
      considerably less than the key lifetime limitation L. On the
      other hand, this approach can be applied in case when messages may
      be lost or reordered (e.g., DTLS records).

6.2. **Constructions that Do Not Require 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 encryption of the data.

6.2.1. **ACPKM Re-keying Mechanisms**

   This section defines periodical key transformation without a master
   key, which is called ACPKM re-keying mechanism. This mechanism can
   be applied to one of the basic encryption modes (CTR and GCM block
   cipher modes) for getting 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 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.

```
                ACPKM          ACPKM                    ACPKM
         K^1 = K ---> K^2 ---...--> K^{l_max-1} ----> K^{l_max}
             |            |                |              |
             |            |                |              |
             v            v                v              v
M^{1}  |==========|==========|  ...  |==========|======:   |
M^{2}  |==========|==========|  ...  |===         |      :   |
   .        .          .         .        .          . :
   :        :          :         :        :          : :
M^{q}  |==========|==========|  ...  |==========|=====  :   |
                   section                            :
                <---------->                      m_max
                   N bit
```
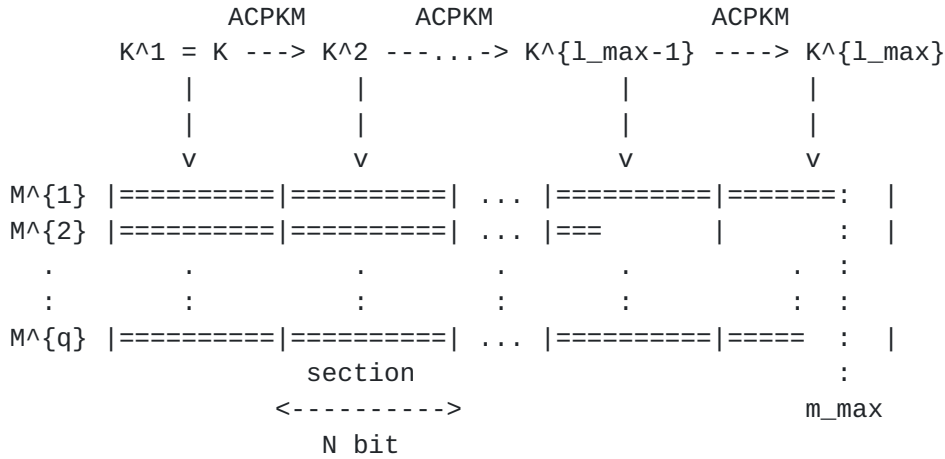
l_max = ceil(m_max/N).

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 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 ACPKM transformation as follows:

    K^{i+1} = ACPKM(K^i) = MSB_k(E_{K^i}(D_1) | ... | 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 | D_2 | ... | 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 )
```

N o t e : The constant D is such that $D_1, \ldots, D_J$ are pairwise different for any allowed n and k values.

N o t e : The constant D is such that the highest bit of its each octet is equal to 1. This condition is important, as in conjunction with message length limitation it allows to prevent collisions of block cipher permutation inputs in cases of key transformation and message processing.

6.2.2. CTR-ACPKM Encryption Mode

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

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

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

o $64 <= n <= 512$;

o $128 <= k <= 512$;

o   the number of bits c in a specific part of the block to be incremented is such that $32 <= c <= 3 / 4 \; n$, c is a multiple of 8;

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

The GCM-ACPKM mode can be considered as the basic 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:

o n in {128, 256};

o 128 <= k <= 512;

o the number of bits c 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;

o authentication tag length t;

o 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 (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 vector IV 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 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 ACPKM-Master re-keying mechanism. This mechanism can
be applied to one of the basic 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 also 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.
_____

                       ACPKM                         ACPKM
             K*_1 = K--------------> K*_2 ---------...---------> K*_l_max
               ___|___                ___|___                    ___|___
              |       |              |       |                  |       |
              v  ...  v              v  ...  v                  v  ...  v
            K[1]      K[t]        K[t+1]    K[2t]      K[(l_max-1)t+1]
K[l_max*t]
              |       |              |       |                  |       |
              |       |              |       |                  |       |
              v       v              v       v                  v       v
M^{1}||=======|...|=======||=======|...|=======||...||=======|...|==    :
||
M^{2}||=======|...|=======||=======|...|=======||...||=======|...|======:
||
 ... ||       |   |       ||       |   |       ||   ||       |   |      :
||
M^{q}||=======|...|=======||====    |...|       ||...||       |...|      :
||
        section                                                        :
      <-------->                                                       :
        N bit
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, that 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
internal ACPKM-Master re-keying mechanism for the periodical key
transformation.

The CTR-ACPKM-Master encryption mode can be considered as the basic
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:

o 64 <= n <= 512;

o 128 <= k <= 512;

o the number of bits c in a specific part of the block to be
   incremented is such that 32 <= c <= 3 / 4 n, c is a multiple of 8;

o 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, |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 internal ACPKM-Master re-keying mechanism for the
   periodical key transformation.

   The GCM-ACPKM-Master authenticated encryption mode can be considered
   as the basic 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:

   o n in {128, 256};

   o 128 <= k <= 512;

   o the number of bits c 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;

   o authentication tag length t;

   o 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 vector IV 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
internal ACPKM-Master re-keying mechanism for the periodical key
transformation.

The CBC-ACPKM-Master encryption mode can be considered as the basic 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:

o $64 <= n <= 512$;

o $128 <= k <= 512$;

o 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 will denote by $D_{\{K\}}$ the decryption function which is a permutation inverse to the $E_{\{K\}}$.

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 each message that is encrypted under
the given initial key does not need to be secret, but must be
unpredictable.

6.3.5. **CFB-ACPKM-Master Encryption Mode**

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

   The CFB-ACPKM-Master encryption mode can be considered as the basic
   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:

   o $64 <= n <= 512$;

   o $128 <= k <= 512$;

   o 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 each message that is encrypted under the given initial key need not to be secret, but 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 internal ACPKM-Master re-keying mechanism
   for the periodical key transformation.

   The OMAC-ACPKM-Master mode can be considered as the basic message
   authentication code calculation mode OMAC, 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:

   o n in {64, 128, 256};

   o 128 <= k <= 512;

   o 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 $|K^j| = k$ and $|K^j\_1| = n$.

   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                            |
|                                                                       |
+-----------------------------------------------------------------------+
```

   Where R_n takes the following values:

   o $n = 64$: $R\_{64} = 0^{59} | 11011$;

o   n = 128: R_{128} = 0^{120} | 10000111;

o   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

Any mechanism described in Section 5 can be used with any mechanism
described in Section 6.

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 some efficient attacks are known for a cipher, it
must not be used. So 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 post-quantum security issues
for symmetric crypto, 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
a size of plaintext transformed on one initial key.

Re-keying can provide backward security only if previous traffic keys
are securely deleted by all parties that have the keys.

9. References

9.1. Normative References

[CMS]   Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
            RFC 5652, DOI 10.17487/RFC5652, September 2009,
            <http://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, <http://www.rfc-editor.org/info/rfc6347>.

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

[GCM]   McGrew, D. and J. Viega, "The Galois/Counter Mode of
            Operation (GCM)", Submission to NIST
            http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/
            gcm/gcm-spec.pdf, January 2004.

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

[NISTSP800-108]
            National Institute of Standards and Technology,
            "Recommendation for Key Derivation Using Pseudorandom
            Functions", NIST Special Publication 800-108, November
            2008, <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>.

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

   [TLS]   Dierks, T. and E. Rescorla, "The Transport Layer Security
               (TLS) Protocol Version 1.2", RFC 5246,
               DOI 10.17487/RFC5246, August 2008,
               <http://www.rfc-editor.org/info/rfc5246>.

   [TLSDraft]
               Rescorla, E., "The Transport Layer Security (TLS) Protocol
               Version 1.3", 2017,
               <https://tools.ietf.org/html/draft-ietf-tls-tls13-20>.

9.2. Informative References

   [AbBell]   Michel Abdalla and Mihir Bellare, "Increasing the Lifetime
               of a Key: A Comparative Analysis of the Security of Re-
               keying Techniques", ASIACRYPT2000, LNCS 1976, pp. 546-559,
               2000.

   [LDC]   Howard M. Heys, "A Tutorial on Linear and Differential
               Cryptanalysis", 2017,
               <http://www.cs.bc.edu/~straubin/crypto2017/heys.pdf>.

   [Sweet32]   Karthikeyan Bhargavan, Gaetan Leurent, "On the Practical
               (In-)Security of 64-bit Block Ciphers: Collision Attacks
               on HTTP over TLS and OpenVPN", Cryptology ePrint
               Archive Report 2016/798, 2016,
               <https://sweet32.info/SWEET32_CCS16.pdf>.

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

Appendix A. Test examples

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 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



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

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

   Plain text P:
   11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88
   00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A
   11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11
   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22
   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33
   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:
   80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F

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

   ACPKM's iteration 1
   Process block 1
```

Input block (CTR):
12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 00

Output block (G):
FD 7E F8 9A D9 7E A4 B8 8D B8 B5 1C 1C 9D 6D D0

Plain text block:
11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88

Cipher text block:
EC 5C CB DE 8C 18 D3 B8 72 56 68 D0 A7 37 F4 58

Process block 2
Input block (CTR):
12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 01

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

Plain text block:
00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A

Cipher text block:
19 89 E7 42 32 62 9D 60 99 7D E2 4B C0 E3 9F B8

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

ACPKM's iteration 2
Process block 1
Input block (CTR):
12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 02

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

Plain text block:
11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00

Cipher text block:
F5 AA BA 0B E3 64 F0 53 EE F0 BC 15 C2 76 4C EA

Process block 2
Input block (CTR):
12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 03

Output block (G):

```
   BC 4F 87 23 DB F0 91 50 DD B4 06 C3 1D A9 7C A4

   Plain text block:
   22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11

   Cipher text block:
   9E 7C C3 76 BD 87 19 C9 77 0F CA 2D E2 A3 7C B5

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

   ACPKM's iteration 3
   Process block 1
   Input block (CTR):
   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 04

   Output block (G):
   68 6F 22 7D 8F B2 9C BD 05 C8 C3 7D 22 FE 3B B7

   Plain text block:
   33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22

   Cipher text block:
   5B 2B 77 1B F8 3A 05 17 BE 04 2D 82 28 FE 2A 95

   Process block 2
   Input block (CTR):
   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 05

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

   Plain text block:
   44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33

   Cipher text block:
   84 4E 9F 08 FD F7 B8 94 4C B7 AA B7 DE 3C 67 B4

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

   ACPKM's iteration 4
   Process block 1
   Input block (CTR):
   12 34 56 78 90 AB CE F0 00 00 00 00 00 00 00 06
```

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

Plain text block:
55 66 77 88 99 AA BB CC EE FF 0A 00 11 22 33 44

Cipher text block:
56 B8 43 FC 32 31 DE 46 D5 AB 14 F8 AC 09 C7 39

Cipher text C:
EC 5C CB DE 8C 18 D3 B8 72 56 68 D0 A7 37 F4 58
19 89 E7 42 32 62 9D 60 99 7D E2 4B C0 E3 9F B8
F5 AA BA 0B E3 64 F0 53 EE F0 BC 15 C2 76 4C EA
9E 7C C3 76 BD 87 19 C9 77 0F CA 2D E2 A3 7C B5
5B 2B 77 1B F8 3A 05 17 BE 04 2D 82 28 FE 2A 95
84 4E 9F 08 FD F7 B8 94 4C B7 AA B7 DE 3C 67 B4
56 B8 43 FC 32 31 DE 46 D5 AB 14 F8 AC 09 C7 39

OMAC-ACPKM-Master mode with AES-256
****
k = 256
n = 128
N = 256
T* = 768

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

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

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

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

K^1|K^1_1 K^2|K^2_1 K^3|K^3_1
9F 10 BB F1 3A 79 FB BD 4A 4C A8 64 C4 90 74 64
39 FE 50 6D 4B 86 9B 21 03 A3 B6 A4 79 28 3C 60
77 91 17 50 E0 D1 77 E5 9A 13 78 2B F1 89 08 D0

```
AB 6B 59 EE 92 49 05 B3 AB C7 A4 E3 69 65 76 C3
9D CC 66 42 0D FF 45 5B 21 F3 93 F0 D4 D6 6E 67
BB 1B 06 0B 87 66 6D 08 7A 9D A7 49 55 C3 5B 48
F2 EE 91 45 6B DC 3D E4 91 2C 87 C3 29 CF 31 A9
2F 20 2E 5A C4 9A 2A 65 31 33 D6 74 8C 4F F9 12
78 21 C7 C7 6C BD 79 63 56 AC F8 8E 69 6A 00 07

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

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

Block number 1
Plain text:
11 22 33 44 55 66 77 00 FF EE DD CC BB AA 99 88

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

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

Block number 2
Plain text:
00 11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A

Input block:
0B B4 AB 8C 11 94 73 35 DB 91 23 CD 6C 10 DB 34

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


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

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

Block number 3
Plain text:
11 22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00
```

```
Input block:
0D 71 EE E7 38 BA 96 9F 74 B5 AF C5 36 95 F9 2C

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

Block number 4
Plain text:
22 33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11

Input block:
6C E7 F8 F3 A8 1A E5 8F 52 D8 49 FD 1F 42 59 64

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


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

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

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

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

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

Block number 5
Plain text:
33 44 55 66 77 88 99 AA BB CC EE FF 0A 00 11 22

Using K1, src doesn't require padding
Input block:
FD E6 71 37 E6 05 2D 8F 94 A1 9D 55 60 E8 0C A4

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

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

Appendix B. Contributors

    o Russ Housley
       Vigil Security, LLC
       housley@vigilsec.com

    o Evgeny Alekseev
       CryptoPro
       alekseev@cryptopro.ru

    o Ekaterina Smyshlyaeva
       CryptoPro
       ess@cryptopro.ru

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

    o Daniel Fox Franke
       Akamai Technologies
       dfoxfranke@gmail.com

    o Lilia Ahmetzyanova
       CryptoPro
       lah@cryptopro.ru

Appendix C. Acknowledgments

    We thank Mihir Bellare, Scott Fluhrer, Dorothy Cooley, Yoav Nir, Jim
    Schaad, Paul Hoffman and Dmitry Belyavsky for their useful comments.

Author's Address

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

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