Network Working Group
INTERNET-DRAFT
Expires: April 2001

Rolf Blom, Ericsson
Elisabetta Carrara, Ericsson
Karl Norrman, Ericsson
Mats Naslund, Ericsson
Sweden

November 15, 2000

RTP Encryption for 3G Networks
<draft-blom-rtp-encrypt-00.txt>

Status of this memo

Abstract

This document describes a method for confidentiality protection (encryption) of the payload in conversational multimedia applications running over the Real-time Transport Protocol [RTP].

The proposal is based on the 3GPP (3rd Generation Partnership Proposal) confidentiality algorithm "f8", and the new Advanced Encryption Standard (AES). The proposed scheme satisfies all the requirements put forward in [CMSec], such as being error-robust and allowing for bandwidth-saving header compression. Most important, the solution is based on a security mechanism that has undergone public scrutiny, and is widely accepted to be secure.

TABLE OF CONTENTS

1. Introduction

As discussed in [CMSec], there are a number of requirements that immediately arise when designing an encryption scheme for packet-switched, real-time data sent on wireless (unreliable) media. The requirements are:

- The encryption scheme should avoid error-propagation (error-robustness)

- The mechanism must be fast, and must be implemented efficiently in thin clients

- The encryption scheme has to show a "fast-forward/rewind" property

- The encryption scheme should not expand the message size

- To allow for header compression over the air link, e.g. [ROHC], it is necessary to avoid end-to-end (e2e) encryption of RTP headers (the security aspects of this are discussed below, in Section 5.)

Therefore, as noted in [CMSec], this leads to the choice of employing e2e encryption only on the RTP payload, using either a block cipher operating in a suitable feedback mode, or a pure stream cipher with a random-access property into different locations of the keystream. We propose a scheme of the former type, and motivate this as follows.

In terms of security, the components of the proposal have undergone a fair amount of public scrutiny without the detection of any weaknesses, see Section 5.

In a feedback type mode, the central ingredient is the block-cipher used. As far as modern cryptology knows, the security basically stands (and falls) with the security of the block cipher if implemented wisely. This means that if a weakness is found, replacing the block cipher with a new one will most likely remedy the security problems.

Good stream ciphers available for public use are rare: there are far more (presumed) secure block ciphers than stream ciphers in this cathegory. In particular, the "random-access" property into the keystream that we desire, disqualifies all but a few stream ciphers such as the Software Encryption Algorithm [SEAL].

Finally, the encryption mechanism (mode of operation) we propose was tailor-made for confidentiality protection in 3G networks.

In section 2, we describe a solution that is used in a similar environment. In section 3, we apply a modified version of this solution in the context of RTP payload encryption, and propose to use the AES algorithm as the cryptographic core of this solution.

1.1. Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" that appear in this document are to be interpreted as described in [RFC-2119].

1.2. Notation

Except when otherwise noted, we use the same nomenclature as in

[CMSec].

The symbol || denotes concatenation of two binary strings, and XOR denotes bitwise addition modulo 2.

The prefix 0x is used to denote hexadecimal numbers. Data and variables are presented with their most significant bytes (or bits) to the left, and bit (and byte) indices are 0,1,2,.. counting left to right. In general, X[i] denotes the ith bit of X, i.e. X = X[0] || X[1] || X[2] ...

2. Background

2.1. 3GPP Confidentiality, the f8-algorithm

To encrypt UMTS data, the 3GPP has developed a solution (see [ES3D]) known as the f8-algorithm. On a high level, the proposed scheme is a variant of Output Feedback Mode (OFB), see [HAC], with a more elaborate initialization and feedback function. As in normal OFB, the core consists of a block cipher. 3GPP specifies this block cipher to be Kasumi, see [ES3D], which is a derivative of Matsui's Misty algorithm, [MAT]. Kasumi may, in principle, be replaced by any secure block cipher (possibly after adjusting some parameters).

In the next sections, we describe f8, Kasumi, its precursor Misty and the recently selected AES.

2.1.1. f8-Mode of Operation

Figure 1 shows the structure of an arbitrary b-bit block size cipher, E, running in what we shall call "f8-mode of operation". (In the 3GPP specifiaction, E is the 64-bit block cipher "Kasumi", see below).

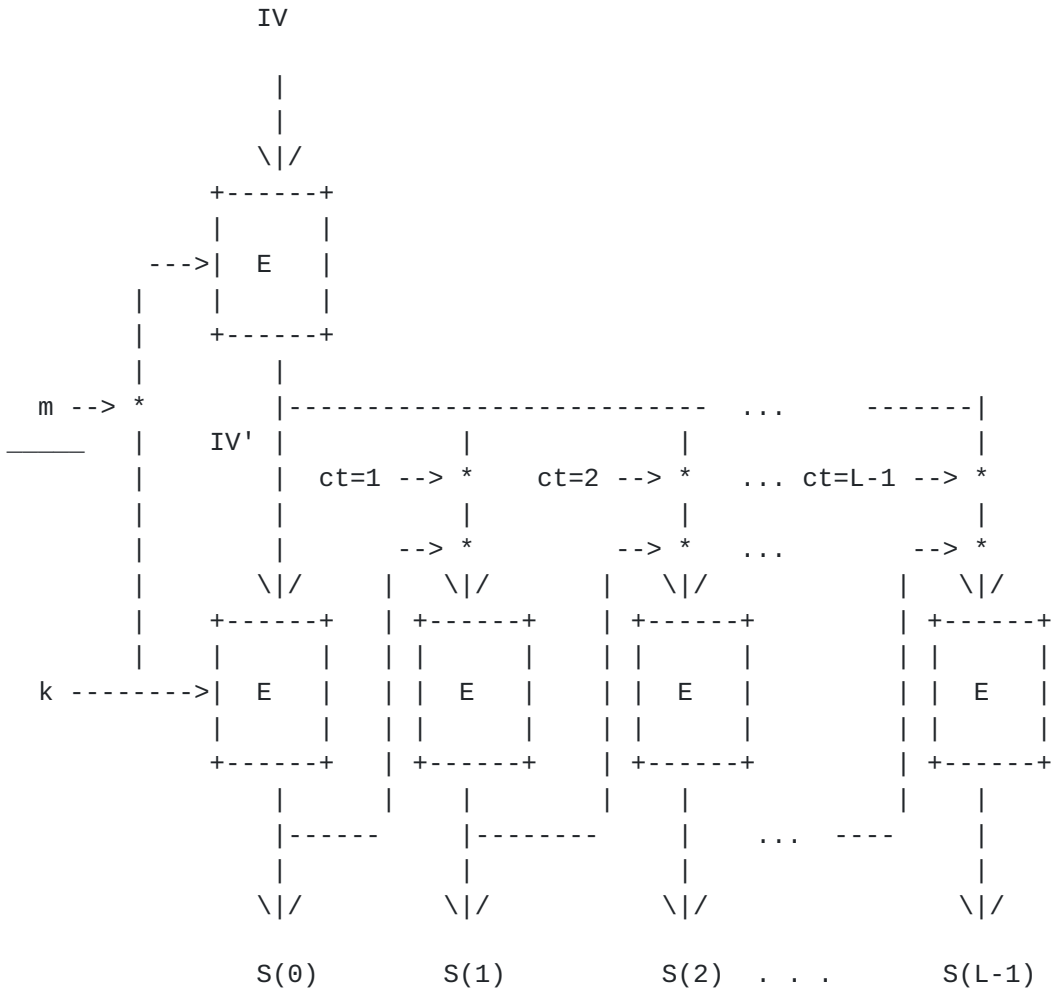

Figure 1. In the figure, asterisk, *, denotes bitwise XOR.

Let E(k,B) be the b-bit output of E when applied to the l-bit key k and plaintext block B. Let ct, IV, IV', and S() denote b-bit quantities, and m be an l-bit string (determined below). The encryption of an n-bit plaintext P = P[0] || P[1] || .. || P[n-1], is then performed as follows.

Set IV' = E(k XOR m, IV), and ct = S(-1) = 00..0. For j = 0,1,.., L-1 where L = n/b(rounded up to nearest integer) compute

S(j) = E(k,IV' XOR ct XOR S(j-1)), (Eq. 1)
ct = ct + 1 mod 2^b (Eq. 2)

Let S = S(0) || S(1) || .. || S(L-1), the concatenation of successive outputs from E. Then, the ciphertext, C, is determined from the plaintext P and this S by

C[i] = S[i] XOR P[i], i = 0,1,..,n-1 (Eq. 3)

i.e. the normal bitwise XOR.

Thus, in the figure, k is the key, and IV is the initialization vector. Notice that the IV is not used directly. Instead it is fed through E under another key to produce an internal, "salted" value (denoted IV'). The motivation for this is explained below. Thus, the input supplied to each consecutive application of E (besides the key k) consists of the salted IV', XORed with the b-bit block-counter, ct, and the previously generated keystream block (if available). The value m is the fixed mask 0x555...(repeated to give as many bits as the key-length). The IV is by 3GPP determined as

IV = COUNT || BEARER || DIR || 00..0

(padded with as many zeros as needed to fill the block length of E). COUNT is a 32-bit value derived from a layer 1 frame counter and a "hyper-frame" counter (to avoid reuse of key stream). BEARER is a 4-bit bearer identifier, and DIR, finally, is a single bit, distinguishing the up-link (terminal to radio base station) from the down-link (base station to terminal).

If the supplied key k is shorter than what is prescribed by E, the key bit-pattern is repeated in the least significant bits to the desired length.

It is clear that the f8-mode of operation has the same properties as normal OFB mode as far as error propagation and synchronization is concerned. From a security point of view, the modified feedback has been included to improve security. Basically, the first application of E to the supplied IV is done to reduce effects of the fact that the IV may be known or perhaps even controlled by an adversary. In fact, even if this is the case, without knowledge of the key, it is hard to predict the internal IV', that is then the initialization vector actually used. Thus, this so called "whitening" or "salting" makes it hard for an adversary to obtain known input-output pairs to E, and makes analysis more difficult. The incorporation of the block counter is there to prevent reuse of keystream (the sequence will not have a short period). (It is assumed by 3GPP that no data of length > 5114 bits are to encrypted, and hence, in practice the block counter will never wrap modulo 2^64. In fact, ct may then be implemented as a 7-bit counter.)

We next discuss Kasumi and its precursor, Misty.

2.1.2. Misty/Kasumi

Misty is a block cipher having block length 64 bits and a 128-bit key. Thus, in OFB-type modes, 192 bits are needed to specify the key and the IV. The design criteria of Misty were speed and, most importantly, provable security against the most common cryptanalytic attacks on block ciphers: linear and differential attacks. See [HAC] for description of these attacks. There are currently no known feasible attacks against Misty.

The differences between Misty and Kasumi are mainly that modifications were made to improve hardware gate count and performance, see [ES3E]. No real security weaknesses have been found, arising from these modifications, again see [ES3E].

2.1.3. Performance

In [ES3D] it is claimed that the f8-design meets the following hardware requirements:

- implementable in hardware in less than 10000 gates (an actual figure of 3000 gates for Kasumi is mentioned)

- at least 2Mb/s encryption speed at 20MHz clock rate

- (re-)initialization within a 10ms time frame

An important thing to note is that most mobile terminals will have the Kasumi based f8-algorithm implemented in hardware, offering very good performance. It should be noted, however, that if we use f8 also for application layer encryption, then we may be competing with the UMTS air-link for the hardware resources. In particular, a fast context switch is needed.

We give some actual performance figures in Section 6.

2.2. AES: an alternative to Kasumi

NIST has recently selected the new Advanced Encryption Standard, AES. This is a block cipher known as "Rijndael" which has a block size of 128 bits, and allows for 128, 192, and 256 bit keys. For further technical details, see [NIST,Ri]. We see AES/Rijndael as an excellent alternative to Kasumi for the cryptographic core of the f8-mode of operation.

- During the last three years, it has undergone extensive cryptologic analysis, without revealing weaknesses

- It is a very fast algorithm (see Section 6)

- Its larger block and key sizes have advantages (discussed later)

3. Proposal for RTP Payload Encryption

With the above discussion in mind, we see an f8-type scheme as the best solution for confidentiality protection of conversational multimedia carried by the RTP protocol.

1) There is no error propagation. Concerning the error rate of the channel, the only assumption is that packets that do arrive at the application layer have no errors in their headers. The error control is typically taken care of by the header compression scheme, [ROHC], and possibly UDP Lite [UDPL]. (Thus, the negligible residual bit error rate over the air link is even further decreased by the header compression.)

2)There is no message expansion since there is no padding. Hence, bandwidth is conserved.

3)As the mechanism is used on the application layer, additional IP level security association numbers and sequence numbers are avoided.

4)Synchronization is easy to achieve via the use of the IV, giving the desired random-access property.

5)It is flexible, as it can be based on any secure block cipher with varying key and block sizes. In particular the AES, Rijndael [Ri] is a good choice. Though slower than a designated stream cipher, speeds of several Mb/s are possible both in hardware and software based on benchmarks done on popular block ciphers, see e.g. [BOS,LIP] and Section 6.

6)It is highly probable that hardware support for Rijndael will exist in terminals.

Next, we discuss the adaptation of the f8-mode of operation for use with RTP.

3.1. F8-Mode for RTP Payload Encryption

Assumptions:

1. We assume that a key exchange/parameter negotiation protocol exists (see also Section 4).

2. For simplicity, we assume the communication to be unicast.

(Extension to multicast is mainly a key management problem and is discussed below.)

3. The transport layer uses UDP. Some modifications to the initialization steps may be needed if some other protocol is used, and we discuss them later.

4. We assume that the application has knowledge of the 16-bit port_number, on which the RTP packets are received. Typically, port numbers are negotiated via a signalling protocol, such as SIP. We furthermore assume that the combination (port_number, SSRC), where SSRC is the 32-bit RTP Synchronization Source, is unique for each media flow belonging to the same multimedia session between the two communicating parties. This must be assured by the signalling protocol during set-up, otherwise security is compromised (see discussion below). Observe that the "key" may then be the same for all media types/flows belonging to the same multimedia session between the two parties, since the distinction responsibility lies on the (port_number, SSRC)-pair.

In case the transport layer does not use UDP, a 16-bit quantity serving as "port_number" and having the uniqueness properties mentioned above must be agreed upon during initial set-up.

The encryption shall be done using the following steps.

3.1.1. Parameter negotiation

Using the key-exchange protocol, the following parameters shall be agreed:

cipher: the algorithm E. The AES algorithm must be supported, other algorithms, with block size at least 64, may be supported.

block_size: almost always determined uniquely by "cipher". It is 64 in the case of Kasumi, and usually 128 for AES (though Rijndael supports larger block sizes) and is therefore normally implicit. Note: as this parameter varies, different numbers of output keystream bits will be generated per application of "cipher", depending on what algorithm is used. This means that a 128-bit block algorithm will be about twice as fast in encrypting a fixed amount of data as a 64-bit algorithm. Notice that since AES must be supported, its definition implies that block_size 128 must be supported in connection with AES.

Note that the internal block counter (this is "ct" in Figure 1) should be implemented as a counter of block_size bits. However, if it is known that the application never generates RTP packets of (payload) size exceeding 2^t times the block size (bits), ct may be implemented as a t-bit counter and then extended with leading zeros to fill the block_size before use. In almost all cases, t = 32 or

even 16 will suffice.

key_size: the length of the key. The default shall be 128.

key: a (pseudo)randomly chosen binary string of key_size bits to be used as the key. Important: we stress that the key must be random or pseudo-random, otherwise security is compromised. For details on pseudo-random generators and guidelines for key generation, see [HAC,FIPS140].

If the key size is not in the set of key lengths supported by the cipher, the key shall be appended to itself as many times as neeeded to obtain, possibly after a final truncation to the most significant bits, a length which is the set of supported key sizes (this may be due to export restrictions). For example, if the minimum key size is 16 bits, the key 0x123 (of length 12) is first turned into 0x123123 (of length 24), and then truncated to 0x1231 which has the desired length.

3.1.2. Cipher Initialization

Given the key and the IV, initialization is performed as in Figure 1, computing the internal IV'. If the cipher allows, pre-computing the cipher's key-schedules (one with the key, and one with key XOR 0x555...) may offer performance enhancement on the actual encryptions that is follow. Thus, it remains to specify how the IV is formed.

3.1.3. IV Calculation

For each RTP packet to be transmitted, the following encryption shall be performed. The payload of each RTP packet (i.e. starting immediately after its RTP header) shall be encrypted by taking information from the RTP header of said RTP packet, and the 16-bit receiving port number (as negotiated inside the signalling protocol), thereby forming the IV:

IV = port_number || SSRC || SEQ || TS || M || PT || 00...0

The SEQ (Sequence Number, 16 bits), SSRC (Synchronization Source, 32 bits), TS (Timestamp, 32 bits), PT (Payload Type, 7 bits), and M (Marker Bit, 1 bit) fields are as specified by [RTP]. If the block_size is between 64 and 104, the IV shall be truncated after the corresponding number of bits in the generic IV (for example, if the block_size is 64, then IV = port_number || SSRC || SEQ). If the block_size is greater than 104, as many zeros as needed to fill the block_size shall be appended.

By this choice of IV and the assumptions on the (port_number,SSRC)-pairs made above, the flow from A -> B will automatically be

encrypted by a keystream distinct from the flow in the opposite direction, B -> A. Some care is thus needed in avoiding reuse of the

same IV inside a multimedia session, in case the latter shares the same key between its RTP sessions. See Sections 4 and 5 for further considerations.

This IV shall then be entered as the IV to the f8-scheme and the internal IV' is computed as described earlier (c.f. Figure 1).

3.1.4. Encryption at sending end

The RTP-payload only (not the RTP-header) shall then be encrypted in accordance with Eq. 1, 2, and 3 above, i.e. be bitwise XORed with the output keystream of the f8-scheme. That is, if the payload length is n, n/"block_size" (rounded up) applications of "cipher" is needed to produce sufficient keystream data. If the packet length is not an integer multiple of block_size, any extra keystream bits generated shall be ignored.

3.1.5. Decryption at receiving end

As the XOR operation is an involution, decryption shall be performed in exactly the same way by the receiver.

3.1.6. Multicast

Observe that in a multicast scenario, each packet belonging to a media flow will be sent to the same port_number for all intended receivers. Thus, as long as the receiving parties share a key with the sender, the above method (IV formation) may be used.

This concludes the formal description of the proposed scheme. Example test vectors of the AES based implementation can be found in Appendix A.

4. Key Management

Though beyond the scope of this draft, we make the following observations that must be taken into account when finding a suitable key-exchange protocol.

First note that as discussed, the IV forming allows us (if desired) to use only one key for the bi-directional flows, and for all media types in the overall multimedia session. Again, we stress that this is due to the fact that we assume that the (port_number/SSRC)-pairs are chosen uniquely during initialization, see above. If not, some media flow may be encrypted by the same keystream as another, and

security is compromised. Distinct keys for different flows may of

course also be used, though efficiency will then be reduced due to frequent context switches.

Key refresh: If a feedback type construction is used (as we propose to do), one must be aware that generating a very long keystream from the same IV/key pair, will eventually lead into "degeneration" of the keystream, making it possible to distinguish it from a random string. Whence, security might be compromised. The maximum allowable length (for a good block cipher) depends only on the block_size, b. Specifically, such bad behaviour will start to occur after about 2^(b/2) iterative applications of the cipher. With b = 128 (for AES) or even b = 64 (Kasumi), this will in practice never happen, as typical packet sizes for, say conversational audio are typically 32 bytes only. (The 3GPP f8-solution was designed with a maximum data length of 5000 bits in mind and thus, a large security margin exists.) If a shorter keystream is generated, there is still a slight chance of having collisions in the keystream. However, the probability of this occurring while encrypting a normal size packet is (with b = 64) on the order 2^(-52) (see [ES3E]).

In [ES3E], it is also advised that the same key should not be used together with more than about 50 million IVs. It is noted that this corresponds to about 40 hours of data at 2Mbit/s. However, in our case, since the sequence number (SEQ) is only 16 bits in length, after that many IVs (packets) the keystream will start to repeat itself. Therefore, if only the sequence number is used (as is the case for a 64-bit block size), the key must be refreshed at least for every 2^16 packets. Some practical implications of this can be examined. Consider for instance an audio stream with 50 packets/sec. The SEQ field will then wrap modulo 2^16 once every 20 minutes or so. If a larger block size is used, then also the 32-bit timestamp (TS) is included in the IV, thereby extending the "effective" sequence numbers to a (theoretical) maximum of 48-bits. This maximum may not always be achieved however. For instance, for an audio application with 20 ms samples, the RTP timestamp typically increments by 160 for each increment in SEQ. The combination SEQ || TS will then have the form (j mod 2^16) || (160j mod 2^32), j = 0,1,.., and hence have period 2^27 (the least common multiple of 2^16 and 2^32/gcd(2^32,160)) rather than 2^48. Still, this is very large (about twice the recommended maximum of 50 million), and the need to refresh the key due to exhaustion of the IV-space should be rare. For other applications, e.g. video, the relation between SEQ and TS may look different.

In summary, assisted by the key management functionality, the application must keep track of when a key-refresh is needed, either due to the fact that the RTP Sequence Number (possibly in combination with the RTP Timestamp) wraps/re-cycles, or because so many IVs (e.g.

> 50 million) have been used with the same key that security may start to be endangered. Obviously, the mechanism must ensure that synchronization of the key refresh is obtained between

sender/receiver. It seems to be a reasonable solution to have the key exchange protocol exchange a "master secret", from which consecutive keys can be derived pseudo-randomly with the master secret as a seed, similar to what is done in [TLS, WTLS].

5. Security Considerations

For general security overview of f8 (and Kasumi), we refer to the analysis in [ES3E] and for the AES algorithm, see [Ri,AES]. (If another block cipher is used, the security of that algorithm will of course be an issue.)

However, there are some specific issues that arise in this application that we discuss in the next paragraphs.

In a real-time scenario, key management must not be overloading, both for time and bandwidth reasons. To use a fixed key for all the RTP sessions which belong to the same multimedia session, special attention has to be placed on possible collisions in the way the IV is formed (see also above). The requirement is not to end up with two identical IVs, for the same key. The RTP standard defines some recommendations on how to use SSRC (unique inside a single RTP session, also accompanied by an anti-collision algorithm, and port numbers are recommended for demultiplexing). However, without special collision detection, there may be unlucky (port_number,SSRC)-combinations. We therefore believe it is up to the implementation not to end up in such a situation, and as noted above, we recommend a careful choice of the receiving ports (being negotiated inside the call control protocol, e.g. SIP). Again, the required property is that port numbers should be chosen to avoid the same combination of SSRC/port number inside RTP sessions which belong to the same multimedia session.

Notice that the uniqueness requirement on the (port_number,SSRC)-combination allows for a theoretical maximum of 2^48 paralell flows, belonging to the same multimedia session.

Security may, of course, also be obtained by asserting that distinct keys are used for the different streams (and their two directions) belonging to the same RTP session if one is willing to perform more frequent context switches.

5.1. Confidentiality of the RTP Payload

It is important to be aware that, as with any stream cipher, the exact length of the payload is revealed by the encryption. This means that it may be possible to deduce certain "formatting bits" of the

payload, as the length of the CODEC output might vary due to certain parameter settings etc. This, in turn, implies that the corresponding

bit of the keystream can be deduced. However, if the stream cipher is secure, knowledge of a few bits of the keystream will not aid an attacker in predicting the following keystream bits. Thus, the payload length (and information deducible from this) will leak, but nothing else.

5.2. Confidentiality of the RTP Header

With our proposal, RTP headers are sent in the clear to allow for header compression. This means that data such as payload type, synchronization source identifier, and timestamp are available to an eavesdropper. Moreover, since RTP allows for future extensions of headers, we cannot foresee what kind of possibly sensitive information might also be "leaked".

Our proposal is a low-cost method, which allows header compression to reduce bandwidth. It is up to the endpoints policies to decide about the security scheme to employ. If the header compression is omitted, other solutions might be applicable, e.g. [IPsec]. In other words, we provide a solution that works in the most demanding scenario: conversational multimedia over low-bandwidth, unreliable media. Of course the solution will then also work in less restricted environments, but we suggest that if one really needs to protect headers, and is allowed to do so by the surrounding environment, then he should also look at alternatives. In addition, we strongly recommend the use of profiles to select the right trade-off for the required level of security.

5.3. Message Integrity

The purpose of this draft is only to treat confidentiality, not integrity. However, we realize the importance of this cryptographic primitive, and a few things can be said.

Adding a "full strength" message authentication code (MAC) to each packet is unacceptable for bandwidth reasons. For instance, a 160-bit HMAC [HAC] field will in principle double the data size for a typical audio packet. If size is reduced to bandwidth-acceptable values by truncating the MAC, perhaps to one or two bytes, a very low security-level is obtained.

Moreover, due to possible errors induced by the transmission medium, an integrity check will put down packets that only have minor errors inflicted "non-adversarialy" by the channel, and will reduce the quality of the received signal.

If bandwidth considerations allow it, adding a standard integrity

mechanism is of course the right thing to do, but again, if we are in

this more favourable situation, one can perhaps use another, more complete security mechanism, e.g. [IPsec].

Recall that we are aiming at providing security for conversational multimedia. What kind of "useful" attacks can be mounted to violate data integrity in such cases?

First, notice that some amount of integrity for the RTP headers is obtained for the parts that enter into the IV-formation (SSRC, SEQ, TS, PT, and M). Namely, should any of these be modified during transmisson, the decryption will only produce "random garbage", reducing an attack to the integrity of these fields to pure denial-of-service (DoS) aspects.

If an attacker records a session for a later replay-attack, possibilities of obtaining something useful seem remote. First of all, we may assume that the replay takes place at a later time within the same session. (Otherwise, with overwhelming probability, the new session will be running with a new distinct key and the receiver would then decrypt with the, for the attacker, wrong key and only "garbage" data is produced). But if the replay occurs within the same session, that would imply that the attacker needs to modify the header sequence numbers and timestamps. As these fields enter the encryption algorithm, an uncorrelated keystream will then be used by the receiver, and garbage will again be produced. We conclude that attempting to do a replay attack, and any attempt in altering important parts of the RTP header will mainly have DoS effects.

In addition, due to the real-time aspects, there will be a natural window mechanism implemented by the application, making alterations of synchronization data (e.g. timestamps) to have mainly DoS aspects.

If a window mechanism and/or buffering for received packets is used, a perhaps more serious attack (also of DoS type), would be to modify the sequence number and timestamp fields, causing the receiver to move his window too far away. The worst case seems to be if this attack is launched at the beginning of the session.

Another attack could be to try to modify bits of the encrypted payload in real-time. Due to the absence of error propagation in the decryption process, it is indeed possible for an attacker to "flip" individual bits for the receiver. However, he does not know the outcome of the flip. Minor alterations (one or two bits) will probably still allow audio/video decoders to reconstruct data at an acceptable quality level for the receiver. However, we cannot exclude the fact that flipping bits will be of some use to an attacker, for instance if those bits have some "simple" meaning to the CODEC. We are aiming at a general solution for RTP data, and one therefore has

to study the format of each possible payload type to understand exactly what can be gained by an attacker in each case. Moreover, we cannot foresee possible future formats, not yet defined.

Again, in most conversational multimedia applications we foresee, we feel that it will be hard to do harmful attacks against the integrity, since data is presented in real-time to the users, and since there is also real-time interaction between the communicating parties. One could say that there is, to some extent, an automatic and "manual" sanity-check done by the users as the data has direct meaning and can be interpreted by them.

In summary, integrity protection may be needed, but it is to a large extent up to the application to decide this, taking bandwidth costs and quality into the budget. The only application-independent observation we can make at this point is that an attacker manipulating the data will be able to launch DoS attacks. However, adding a MAC field does not prevent this, it merely provides a detection method before data enters the CODEC.

6. Implementation experience and simulation results

We have implemented the f8-mode of operation using the AES as the underlying block cipher with 128 bit block size and 128 bit key size. Both algorithms were implemented in C (using the Microsoft Visual C++ Compiler).

The implementation of AES was made compact, that is, we only used tables for the S-box, its inverse and the round constants, for a total of 552 bytes. Everything else was calculated in real-time. With this approach we were able to encrypt at approximatly 22 Mbit/s on a Pentium II with a clock frequency of 266 Mhz.

When measuring the encryption-speed of the f8-algorithm we used simulated RTP-packets with a typical 96-bit header (which was not encrypted) and a 32, 64 and 128 octet random payload (i.e 256, 512 and 1024 bits). With this scenario we need one block encryption to set the internal IV, and two, four and eight block encryptions respectively, to produce the keystream for each packet. Heuristicly, this implies that the f8-algorithm should be able to encrypt somewhere around 16.5 Mbit/s for 32-byte packets. The actual throughput will be a bit less because of some overhead. Since encryption and decryption are the same operation, the same figures hold for decryption.

We encrypted 10^6 packets of each size and took the average encryption times, which are presented in the table below.

Payload Size (bits)	Encrypt/Decrypt (Mbit/s)
256	13.6
512	15.9
1024	17.9

Table 1: Performance of AES-based f8.

Calculation of the internal IV (copying of some data from the packet header and one block encryption) takes around 6 microseconds.

Key-refresh and initialization of the keystream generator (we view initialization as a key-refresh from a non-existent key) takes approximatly 12 microseconds.

For comparison, 20Mb/s at 100MHz is reported for Misty in [MIT]. This is about 3 times faster than typical (optimized) 3DES implementations.

Estimates for hardware performance for AES can be obtained from [AES,Ri]. Similar estimates for Kasumi and f8 can be found in [ES3E].

7. Conclusions

We have made a proposal for a secure, fast, and flexible encryption scheme that has the necessary robustness properties in terms of fault-tolerance needed in conversational multimedia applications over low-bandwidth, unreliable media.

8. Intellectual Property Rights Statement

Pursuant to the provisions of [RFC-2026], the authors represent that they have disclosed the existence of any proprietary or intellectual property rights in the contribution that are reasonably and personally known to the authors. The authors do not represent that they personally know of all potentially pertinent proprietary and intellectual property rights owned or claimed by the organizations they represent or third parties.

9. Acknowledgments

We thank Krister Svanbro and Vicknesan Ayadurai for instructing us on the inner workings of header compression, and Andras Mehes and Jari

Arkko for helpful comments on this draft. We are grateful to Morgan

Lindqvist, Johan Sjoberg, Torbjorn Einarsson, and Magnus Westerlund
for sorting out various RTP questions.

10. Authors' addresses

Rolf Blom Tel: +46 8 58531707
Ericsson Research
Stockholm, Sweden EMail: rolf.blom@era.ericsson.se

Elisabetta Carrara Tel: +46 8 50877040
Ericsson Research
Stockholm, Sweden EMail: elisabetta.carrara@era.ericsson.se

Karl Norrman Tel: +46 8 58531225
Ericsson Research
Stockholm, Sweden EMail: karl.norrman@era.ericsson.se

Mats Naslund Tel: +46 8 58533739
Ericsson Research
Stockholm, Sweden EMail: mats.naslund@era.ericsson.se

11. References

[AES] NIST, "Advanced Encryption Standard (AES)",
http://csrc.nist.gov/encryption/aes/

[BOS] Bosselaers, A., "Fast Implementations on the Pentium",
http://www.esat.kuleuven.ac.be/~bosselae/fast.html

[ES3D] ETSI SAGE 3GPP Standard Algorithms Task Force, "Security Algorithms Group of Experts (SAGE); General Report on the Design, Specification and Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms", Public report, Draft Version 1.0, Dec 1999.

[ES3E] ETSI SAGE 3GPP Standard Algorithms Task Force, "Security Algorithms Group of Experts (SAGE) Report on the Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms", Public report, Draft Version 1.0, Dec 1999.

[FIPS140] NIST, "Security Requirements for Cryptographic Modules", FIPS PUB 140-1

[HAC] Menezes, A., Van Oorschot, P., and Vanstone, S., "Handbook of

Applied Cryptography", CRC Press, 1997, ISBN 0-8493-8523-7.

[IPsec] McGrew, D., Fluhrer, S., Peyravian, M., "The Stream Cipher Encapsulating Security Payload", Internet Draft, July 2000

[LIP] Lipmaa, H., "AES Ciphers: speed", http://www.tml.hut.fi/~helger/aes/

[MAT] Matsui, M., "New Block Encryption Algorthm MISTY". In Eli Biham (Ed.): Fast Software Encryption, 4th International Workshop, FSE '97. Proceedings. Lecture Notes in Computer Science, Vol. 1267, Springer-Verlag 1997, pp. 54-68.

[MIT] Mistubishi Electric, "MISTY", http://www.mitsubishi.com/ghp_japan/misty/

[RFC-2026] Bradner, S., "The Internet Standards Process -- Revision 3", RFC2026, October 1996.

[RFC-2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC-2119, March 1997.

[Ri] Daemen, J. and Rijmen, V.: "AES Proposal: Rijndael", available at http://www.esat.kuleuven.ac.be/~rijmen/rijndael

[ROHC] Burmeister, C., Clanton, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Zheng, H., "RObust Header Compression (ROHC)", Internet Draft, October 2000

[RTP] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V.: "RTP: a Transport Protocol for Real-Time Applications", RFC 1889, Jan. 1996.

[CMSec] Blom, R., Carrara, E., and Naslund, M.: "conversational Multimedia Security in 3G Networks", IETF Draft, November 2000.

[SEAL] Rogaway, P. and Coppersmith, D.: "A Software-Optimized Encryption Algorithm", Journal of Cryptology, vol 11(4), 1998, 273-287.

[TLS] Dierks, T., Allen, C., "The TLS Protocol", RFC 1998, Internet Draft, November 12

[WTLS] Wireless Application Forum: "WAP WTLS, Wireless Application Protocol Wireless Transport Layer Security Specification", Version 18-Feb-2000.

Appendix A. Example Test-vectors

Here is an example of the intermediate values during an encryption using the AES algorithm in f8-mode.

The data encrypted is a single RTP-packet with a 256-bit (pseudo-randomly generated) payload. All values are in hex. Refer to Figure 1 for notation.

key:
  000102030405060708090a0b0c0d0e0f

key XORed with 555... :
  55545756515053525d5c5f5e59585b5a

Rijndael-internal expanded key:
  00010203 04050607 08090a0b 0c0d0e0f
  d6aa74fd d2af72fa daa678f1 d6ab76fe
  b692cf0b 643dbdf1 be9bc500 6830b3fe
  b6ff744e d2c2c9bf 6c590cbf 0469bf41
  47f7f7bc 95353e03 f96c32bc fd058dfd
  3caaa3e8 a99f9deb 50f3af57 adf622aa
  5e390f7d f7a69296 a7553dc1 0aa31f6b
  14f9701a e35fe28c 440adf4d 4ea9c026
  47438735 a41c65b9 e016baf4 aebf7ad2
  549932d1 f0855768 1093ed9c be2c974e
  13111d7f e3944a17 f307a78b 4d2b30c5

Rijndael-internal expanded value of (key XOR 555_):
  55545756 51505352 5d5c5f5e 59585b5a
  3e6de99d 6f3dbacf 3261e591 6b39becb
  2ec3f6e2 41fe4c2d 739fa9bc 18a61777
  0e33034f 4fcd4f62 3c52e6de 24f4f1a9
  b992d079 f65f9f1b ca0d79c5 eef9886c
  30568051 c6091f4a 0c04668f e2fdeee3
  447e91c9 82778e83 8e73e80c 6c8e06ef
  1d114e99 9f66c01a 11152816 7d9b2ef9
  8920d766 1646177c 07533f6a 7ac81193
  7aa20bbc 6ce41cc0 6bb723aa 117f3239
  9e81193e f26505fe 99d22654 88ad146d

RTP-packet header fields:
  version   = 2
  padding   = 0
  extension   = 0
  CSRC count   = 0
  marker bit   = 0

```
     payload type = 0
     sequence no. = 3e7a
```

```
  timestamp    = 7e5d40a7
  SSRC         = 7b777a8f

Receiver port number:
  abcd


IV:
  abcd7b777a8f3e7a7e5d40a7003e0000

IV':
  7f25578863921e41120b09ebfdd43f1c


Encryption of bits 0 to 127 of the payload

ct: 0
S(-1)                 : 00000000000000000000000000000000
S(-1) XOR IV'         : 7f25578863921e41120b09ebfdd43f1c
plain text P[0..127]  : 9979b51c83ac87a3330a6178cc3b1aa6
final keystream S(0)  : b57d2b3337b281f04645bed7082af95d
cipher text C[0..127] : 2c049e2fb41e0653754fdfafc411e3fb


Encryption of bits 128 to 255 of the payload

ct: 1
S(0)                   : b57d2b3337b281f04645bed7082af95c
S(0) XOR IV'           : ca587cbb54209fb1544eb73cf5fec640
plain text P[128..255] : 79cddb405384255385f11619ad46e86f
final keystream S(1)   : e7d836679304e5c7f0881b067e3d682c
cipher text C[128..255]: 9e15ed27c080c09475790d1fd37b8043


This Internet-Draft expires in April 2001.
```