The Eris hybrid cipher

Sandy Harris
sandy@coldstream.ca
sandyinchina@gmail.com

February 2010

Abstract:

An earlier paper by the same author suggested combining a block cipher and a stream cipher to get a strong hybrid cipher. This paper proposes a specific cipher based on those ideas, using the HC-128 stream cipher and a tweakable block cipher based on Serpent.

Table of Contents

The Eris hybrid cipher....1
- Abstract: 1
- Overview....3
- Components....3
  - The stream cipher....4
  - The block cipher core....4
    - Alternate Serpent-based hybrids 4
  - The outer layer....5
    - Round key mixing....6
    - The tweaked transform....6
  - Key schedule....6
    - Dual key applications....7
- Analysis....8
  - Re-keying....8
  - Algebraic attacks 9
  - Linear and differential attacks....9
  - Weak keys....9
  - Novel attacks....10

Overview

In a previous paper [i] I suggested using a stream cipher and a block cipher together to derive a cipher that is, in some ways, stronger than either. In this paper I work out one such design in detail.

The basic technique of using a stream cipher and a block cipher together is applicable to any two ciphers. It is intended to make it difficult to attack either of the underlying ciphers:

- the stream cipher, because the attacker cannot get keystream data without first at least partially compromising the block cipher.
- the block cipher, because some stream cipher output is mixed in for every block so the attacker cannot get data for multiple blocks encrypted in exactly the same way

In the previous paper, I used AES-128 and RC4-128 and compared the combination to AES-256. This let me concentrate on the basics of the combining techniques and on the properties of hybrid ciphers in general.

Here, the focus shifts. I propose a specific cipher, and work out the details of the combining techniques and of the choice of ciphers.

- The stream cipher is HC-128 [ii].
- The block cipher is Serpent [iii].

The proposed cipher is Eris, named for the Goddess of Confusion.

Components

The overall design has three components. The **stream cipher** provides the round keys and tweaks for a **tweakable block cipher** [iv] . The block cipher is split into two parts, a **cryptographic core**, and an **outer layer**. Later sections give details for all three parts.

The notion of splitting a block cipher into core and outer layer comes from the MARS paper:

> Two decades of experience in cryptanalysis has taught us that different parts in a cipher play very different roles in assuring the security of the cipher. In particular, it appears that the top and bottom rounds in the cipher usually have a different role than the middle rounds in protecting against cryptanalytical attacks.
>
> We therefore designed MARS using a mixed structure, where the top and bottom rounds are designed differently than the middle ones. [v]

Eris goes further in this direction than MARS, adding a third radically different component, the stream cipher. In Eris, the outer layer of the block cipher not only mixes in additional key material and provides a great deal of diffusion but also uses stream cipher ouput to tweak the block cipher.

Partitioning the design in this way allows some components – the stream cipher and the core of the block cipher – to use existing designs so all the existing analysis of them applies. The stream cipher is HC-128. The core of the block cipher is 24-round Serpent. Only the outer layers of the block cipher are newly designed components.

The stream cipher

A major consideration in choosing the stream cipher is speed, though of course security is also an issue. It is also convenient to have a stream cipher that produces output in large chunks, 32 or 64 bits per iteration, since we will be using it in large chunks.

RC-4 is an obvious possibility, perhaps the most widely used of stream ciphers, but it produces output a byte at a time and some weaknesses have been found. Also, it is an older design; newer ciphers such as some of the Estream [vi] candidates are faster and claim to be at least as secure.

Py (Roo) [vii] would be a reasonable choice. Py was among the fastest Estream candidates, and it provides large output chunks. Some attacks have been published but they require large samples of keystream, well beyond the 2^{64} bytes the cipher was designed to be secure for, so the designers argue they are not real attacks. Whatever the theorectical merits of those attacks, they almost certainly are not dangerous in a hybrid cipher application, since an atacker cannot get *any* keystream material without first compromising the block cipher.

There are also variants of Py designed to block some of the attacks, TPy and the RCR ciphers; any of these might be used as well. So might any of the other Estream software portolio ciphers.

HC-128 [viii] seems a sensible choice. It has survived considerable analysis, going through the whole Estream process to become one of the final protfolio ciphers. It is almost as fast as Py and produces its output in 32-bit chunks.

The block cipher core

The block cipher's overall design is based on suggestions in the MARS papers, There is an outer layer whose design emphasises diffusion surrounding a cryptographic core which is a more conventional block cipher.

In Eris, the core is 24-round Serpent. The Serpent design document includes analysis showing that “16- round Serpent would be as secure as Triple DES” [ix], but the authors used 32 rounds in their Serpent specification because they wanted a large safety factor. In Eris, other components provide safety factors, so 24 Serpent rounds appears to be enough.

The only other change to Serpent in this usage is that Eris with a 256-bit key does not use the Serpent key schedule; instead the stream cipher generates the round keys. For larger keys, the Serpent schedule is used but the result is XOR'd with stream cipher output.

Alternate Serpent-based hybrids

Of course my proposal is by no means the only way to use Serpent in a hybrid cipher. A more conservative design would dispense with my outer layer and just construct a tweakable version of full 32-round Serpent, then drive the tweaks with stream cipher output. This is quite easily done.

A general method, applicable to any block cipher, is to apply whitening with the stream cipher ouput used to alter the whitening every round or every few rounds.

It would also be possible to apply a tweak at every round of Serpent. The eight Serpent S-boxes each have an inverse so there are sixteen S-boxes overall. Use four bits of key per round to select among those and you get a 32-round cipher that takes a 128-bit tweak. Drive that tweak with a stream cipher and you get a block cipher immune to algebraic attack.

As a thought experiment, consider using Serpent as both the stream cipher and the block cipher. Run 16-round Serpent in counter mode as the stream cipher and whitened 16-round Serpent as the block cipher. Use output from the counter instance to alter the whitening for each round of the other instance. This has almost exactly the overheads of standard 32-round Serpent; the only extra work is changing the whitening data, and that is not an expensive operation.

It is of course not clear that such a construction is as secure as, let alone more secure than, straight 32-round Serpent. However, it does have some interesting properties, as would any hybrid cipher. It is resistant to algebraic analysis and to any attack on the block cipher that requires many blocks encrypted in exactly the same way, which includes (at least the usual forms of) both linear and differential cryptanalysis.

I note in passing that while splitting Serpent into two 16-round chunks to build a hybrid cipher might be reasonable, the technique is distinctly less plausible for other ciphers. AES-256, for example, has 14 rounds, but using two 7-round AES instances in this construction would likely not be a good idea. One might try two instances with ten rounds each, as for AES-128, but then overheads would be significantly higher.

The significant point for our discussion is that if the construction using two instances of 16-round Serpent is secure (not clear, but not entirely implausible) and HC-128 is at least as secure as Serpent 16 in counter mode (plausible, even likely), then the Eris construction, even with only 16 rounds of Serpent in the core instead of 24, is also secure.

The outer layer

The main design goals for the outer layer are

- to mix in a significant amount of additional key material
- to provide a great deal of diffusion
- to provide tweakability.

The MARS paper describes their design goals for this layer:

> Many cryptanalytical techniques (including linear and differential cryptanalysis) treat the top and bottom rounds of the cipher differently than the middle rounds. Typically, these techniques begin by guessing several key bits, hence “stripping out” some of the top/bottom rounds of the cipher, and then mounting the cryptanalytical attack against the remaining rounds. This suggests that the top and bottom rounds of the cipher play a different role than the middle rounds in protecting against cryptanalytical attacks. Specifically, for these rounds we care more about fast avalanche of the key bits (which is a combinatorial property) than about resistance to cryptanalysis. Theoretical evidence for the different role played by the top and bottom rounds can be found in the Naor-Reingold constructions [11], in which a “cryptographic core” is wrapped with some non- cryptographic mixing.
>
> Therefore, in the design of MARS the middle rounds are viewed as the “cryptographic core” and are designed differently than the top and bottom rounds, which are viewed as “wrapper layers”. Specifically, the wrapper layers consist of first adding in key words, and then performing several rounds of (unkeyed) S-box based mixing, providing rapid avalanche of key bits. [x]

For Eris, making the cipher tweakable and driving the tweaks with stream cipher output are additional design goals. A 128-bit tweak is required to get immunity from algebraic attacks.

Eris therefore uses 64 bits of tweak in the input transform and 64 on the output side. The two transforms are mirror images of each other – exactly the same operations, but in reverse order – so I

describe only the input transform.

The input transform has four rounds, each consisting of four steps. The first two steps mix in a 128-bit round key. The other two steps apply 16 bits of tweak to control additional mixing.

Round key mixing

The round key mixing operation has two steps.

- First the 128-bit text and 128-bit round key are combined using unsigned addition mod 2^{32}.
- Then the column mixing transform from Whirpool is applied to the whole block.

Addition is chosen as the mixing operation because it is fast and, thanks to carries, it gives marginally better diffusion than XOR. Also, XOR is used in several block cipher modes, in the Serpent round function, and in the Whirlpool transform, so using something different here makes sense.

The Whirlpool operation operates on 128 bits considered as a vector of four 32-bit words. It makes each output word depend on all input words. In this application, that also makes each output word depend on all words of the round key. This contributes to the required fast avalanche.

The tweaked transform

The tweaked transform is based on AES [xi] and uses 16 bits of tweak. As in AES, the 128-bit block is considered as a 4x4 array of bytes and there are two steps:

- First a rotation is applied to the array rows
- Then a transform is applied to the array columns.

The Eris column transform is identical to that used in AES. It operates on the columns of the array, making each output byte in the column depend on all input bytes. The operation is based on matrix multiplication in a finite field.

The AES row transform is simple; three rows are shifted by fixed numbers of bytes and one is left unchanged. For Eris, we use a more complex transform to introduce the tweak.

The Eris row transform is key-dependent rotations of the 32-bit words of the rows. Each word rotation uses 4 bits of tweak; for tweak *t*, the word is left-rotated *2t+1* bits, so possible rotations are 1,3, 5..31 bits. This is more convenient than using 5 bits per word and it ensures some rotation even for *t=0*.

This is stronger than the row transform in AES in several ways – it is applied to all rows rather than just three, it operates at bit level rather than bytes, and it is key-dependent. However, it is also similar to the AES transform in that the basic operation is rotating rows; if the AES row transform works well with the AES column transform, then this should as well.

Key schedule

The recommended key size for general use of Eris is 256 bits. 128-bit keys are supported for backward compaibility, and keys of 384, 448 or 512 bits are also supported.

Eris always uses 32 128-bit round keys. Twenty-four are used by the Serpent core, the other eight by the outer layer. These round keys always depend on stream cipher output. For 128-bit or 256-bit keys, they depend only on that, but for larger keys the Serpent key schedule is in play as well.

Eris also uses 128 bits of tweak per round, generated by the stream cipher.

HC-128 takes a 128-bit key and 128-bit initialisation vector. Eris normally takes a 256-bit key, using half as the HC-128 key and half as the IV. Key scheduling has three steps.

- First the stream cipher does its own key schedule with the 128-bit key and 128-bit IV.
- Second, stream cipher output is used to generate the set of 32 static round keys – 24 128-bit round keys for Serpent and eight more 128-bit sub-keys to be mixed in by the outer layer part of the bock cipher.
- Thirdly, during actual encryption, stream cipher output provides a tweak for each block.

With a 128-bit key, use that as the HC-128 key. The IV, in succesive 32-bit words, is hexadecimal: d1310ba6, 98dfb5ac, 2ffd72db, d01adfb7. These are the first 128 bits of an expansion of pi, from Blowfish [xii].

Keys of 384, 448 or 512 bits are also supported:

- Give 128, 192 or 256 bits to Serpent and let its key schedule create 32 round keys
- Then give the other 256 to HC-128
- XOR stream cipher output into the round keys.

This is secure if *either* the Serpent key schedule *or* HC-128 is.

Dual key applications

In some applications, one might want a cipher specific to an organisation or a group of correspondents.

Schneier mentions the possibility for the GOST cipher [xiii]; any organisation can generate its own S-boxes. For CAST-128 [xiv] or CAST-256 [xv], an organisation could generate its own S-boxes by following the Mister and Adams paper [xvi]. It could be done for Serpent by generating a new set of S-boxes following methods described in the Serpent paper.

A criticism is that trying to keep the S-boxes secret violates Kerckhoffs' Principle [xvii], and if they are not secret, then the cipher is no stronger than the original. Another is that generating new S-boxes for each group of users is too much work.

With Eris, we can get organisation-specific behaviour from the cipher without needing to generate new S-boxes. Use two 256-bit keys, an organisational key K_O and a session key K_s. Run the Serpent key schedule with K_O then load K_s into HC-128 and XOR stream cipher output into the round keys.

The objection based on Kerckhoffs' Principle does not apply. Losing either key is not a complete disaster; an enemy who knows K_O still faces Eris-256 and one who gets K_s faces 24-round Serpent. In both cases, the use of a second key complicates almost any attack somewhat, even if that key is known.

If this approach is used, K_O should be subject to all the normal procedures of key management. In particular, the key should be changed both on some regular schedule and whenever outside conditions – such as a suspected compromise or someone leaving the group – warrant it. It need not change nearly as often as K_s. but it *must* change sometimes.

Analysis

As for any cipher, the design goal for Eris is to be secure against all attacks, but no-one should seriously think it is secure against any until extensive anaysis has been done by people other than the designer. That said, there are some reasons to suppose it might be secure.

First, of course, it is based on Serpent. I know of no analysis specifically of 24-round Serpent, but full 32-round Serpent has been extensively analysed and not found wanting. The Serpent paper says 16 rounds "are sufficient to block all currently known shortcut attacks". Several papers have proposed attack on reduced-round variants [xviii] [xix] [xx], but none I am aware of go above 11 rounds.

Also, the proposed reduced-round attacks all require at least 2^{100} known plaintexts encrypted with a single key. The hybrid cipher, which denies the attacker even a pair of such texts, therefore makes them utterly irrelevant.

HC-128 has also undergone extensive analysis as part of the Estream project, with no weaknesses found.

The aspects of Eris that really need analysis – because they are the most novel and therefore both the most likely to be flawed and in some ways the most interesting – are the basic notion of a hybrid cipher and the detailed design of the outer layers.

Re-keying

The theoretical limits of Eris sceurity appear to be high, at least 2^{64} bytes.

HC-128 is designed to give a secure keystream for up to 2^{64} bytes. In a hybrid cipher, the keystream is less accessible to the attacker; he must at least partially compromise the block cipher to get any keystream data. It therefore seems possible that in this application, HC-128 remains secure beyond 2^{64} bytes.

For a non-hybrid block cipher with 128-bit blocks, a codebook attack begins to do damage at 2^{64} blocks. Linear and differential attacks generally require even more data than that. For example, all of the attacks against reduced round Serpent cited above require at least 2^{100} blocks. Hybrid ciphers are designed to resist these attacks; the attacks all require many blocks encrypted the same way and the hybrid does not provide those. It therefore seems possible that the Eris hybrid cipher remains theoretically secure beyond 2^{64} blocks.

Against an ideal cipher with 128-bit blocks, brute force takes 2^{127} computation and one known plaintext while a complete codebook needs 2^{128} blocks of data and one lookup. In either case, the computation*storage product is near 2^{128}. The design goal for Eris is that all possible attacks have a product above 2^{127}. That said, an attack could be considered a theoretical "break" if both computation and plaintexts are under 2^{128}, and either is significantly under.

In practice, however, the limits are much lower than in theory. Any sane user or protocol will rekey long before the limits of the cipher are approached, or before an enemy can collect enough data for linear or differential analysis.

For example, the Yarrow paper [xxi], on using a block cipher in counter mode as a random number generator, suggests $2^{n/3}$ blocks as an upper limit on the security parameter that controls rekeying. In normal use, the parameter is set to some lower value, generally much lower.

As another example, IPsec [xxii] has a safety mechanism that rekeys after 2^{32} packets even if the network administrator fails to set criteria for more frequent rekeying. In normal use, rekeying is more frequent, generally much more frequent.

For Eris, take 2^{40} blocks as a safe upper limit. This should not be an onerous requirement; most applications rekey long before that anyway.

Algebraic attacks

During actual encryption, 128 bits of stream cipher output are used per block, controlling transforms in the outer layer part of the block cipher. This makes the block cipher immune to algebraic attack, as explained in my previous paper.

Suppose the algebraicist has, for an *n*-bit cipher, *n* equations expressing the output bits in *n+k* variables, *n* input bits and *k* key bits. (*k* might be either the number of bits in the primary key or the total in all the round keys; it does not matter for this argument.) Each known plaintext/ciphertext gives him another set of such equations with different input and output bits but the same *k* key bits. With many (*m*) such pairs he gets *mn* equations in *k* variables. For linear equations, this becomes soluble when *mn >= k.* Even with non-linear equations, it may become soluble eventually.

However, if we mix *s* bits of stream cipher input in for each block, then each plaintext/ciphertext gives *n* new equations, but also *s* new variables. With *m* pairs, the attacker has *mn* equations in *k+ms* variables. Even if the equations are linear, he needs *mn >= k+ms* before he gets a soluble system. With *s = n,* this never happens. *mn >= k+ms and n=s* reduces to *0 >= k* which is not possible unless the basic cipher design is spectacularly bone-headed.

In Eris, the outer layer mixes in 128 tweak bits per block, 64 before and 64 after the core encryption. This meets the s=n criterion, so all algebraic attacks are blocked.

Courtois and Pieprzyk [xxiii] have suggested that Serpent might have a vulnerability to a form of algebraic attack because its S-boxes are rather small. The claim is controversial; I do not intend to enter that discussion. However, the hybrid cipher construction using enough stream cipher output blocks *any* algebraic attack, including the Courtois and Pieprzyk proposal.

Linear and differential attacks

Two very powerful general-purpose attacks against block ciphers are linear and differential cryptanalysis. The Serpent paper includes analysis showing that neither should be expected to succeed against Serpent.

However, the hybrid cipher makes at least the usual forms of these attacks *completely* inapplicable. Both require large samples of data encrypted with the same key. If the whitening changes with every round, such samples become unobtainable. Normal Serpent is *resistant* to these attacks, but the hybrid is completely *immune*.

Of course it might be possible to construct variants of the attacks for attacking hybrid ciphers, but straightforward differential or linear cryptanalysis simply does not work.

Weak keys

When Eris is used with a 128-bit or 256-bit key, the HC-128 key schedule is used and HC-128 generates the round keys for both Serpent and the outer layers. This is as secure as HC-128 against the risk of weak keys.

With keys of 384, 448 or 512 bits, both the Serpent key schedule and HC-128 are used in generating round keys; first the Serpent key schedule is run, then stream cipher output is XORed in.

In these cases, the cipher is resistant to attacks which depend on finding a class of weak keys. No weak keys are currently known for either Serpent or HC-128, so this is not of great concern, but it is a nice property of the combination. Finding weak keys for either Serpent or HC-128 does an attacker no good since the actual round keys have the other method XORed in as well.

Even an attacker who finds a class of weak keys for both ciphers has unfavorable math to contend with. Suppose one in 2^n Serpent keys is weak and one in 2^m for HC-128. Only one in 2^{mn} combined keys will then show both weaknesses. For any plausible values of m and n, the combination is quite safe. The attacker needs either serious weaknesses in both ciphers (m and n are both reasonably small) or a horrendous flaw in one (m or n is tiny) before an attack on the combination is feasible.

Novel attacks

The MARS paper also has:

> Another advantage of this mixed structure is that it is likely to provide better resistance against new (yet undiscovered) cryptanalytical techniques. Namely, a cipher consisting of two radically different structures is more likely to be resilient to new attacks than a homogeneous cipher, since in order to take advantage of a weakness in one structure one has to propagate this weakness through the other structure. Viewed in this light, the mixed structure can be thought of as an "insurance policy" to protect the cipher against future advances in cryptanalytical techniques. [xxiv]

Eris shares the structure, and has the stream cipher as a third radically different element, so it should have this advantage as well.

i Sandy Harris **Exploring Cipherspace: Combining stream ciphers and block ciphers** http://eprint.iacr.org/2008/473
ii Hongjun Wu, **Stream Cipher HC-128**, submitted to the Estream project, http://www.ecrypt.eu.org/stream/hcp3.html
iii Ross Anderson, Eli Biham & Lars Knudsen, **Serpent: A Fleible Block Cipher With Maximum Assurance**, first AES Candidate Conference, 1999, http://www.cl.cam.ac.uk/~rja14/serpent.html
iv M. Liskov, R. Rivest, and D. Wagner, **Tweakable Block Ciphers**, Crypto 2002 http://www.eecs.berkeley.edu/~daw/papers/tweak-crypto02.pdf
v Burwick, Coppersmith et al. **MARS - a candidate cipher for AES**, IBM, 1999 http://domino.research.ibm.com/comm/research_projects.nsf/pages/security.mars.html
vi eSTREAM, the ECRYPT Stream Cipher Project, http://www.ecrypt.eu.org/stream/
vii Eli Biham, Jennifer Seberry, **Py (Roo,): A Fast and Secure Stream Cipher using Rolling Arrays**, submitted to the eSTREAM project, 2005. http://works.bepress.com/jseberry/93/
viii **HC-128**, as cited above
ix **Serpent**, as cited above
x **MARS** , as cited above
xi National Institute Of Standards and Technology AES links; http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html
xii Bruce Schneier, and Mike Schaudies, hex digits of pi, http://www.schneier.com/code/constants.txt
xiii Bruce Schneier, Applied Cryptography, discussion of GOST cipher, p 331-334
xiv Carlisle Adams, **The CAST-128 Encryption Algorithm**, IETF RFC 2144 http://www.ietf.org/rfc/rfc2144.txt
xv Carlisle Adams & J Gilchrist, **The CAST-256 Encryption Algorithm**, IETF RFC 2612 http://www.ietf.org/rfc/rfc2612.txt
xvi Serge Mister & Carlisle Adams, **Practical S-Box Design**, SAC 96, http://adonis.ee.queensu.ca:8000/sac/sac96/papers/paper7.ps
xvii Auguste Kerckhoffs, La Cryptographie Militaire, 1883 http://petitcolas.net/fabien/kerckhoffs/
xviii J Kelsey, T Kohno, B Schneier, **Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent**, FSE 2001
xix Eli Biham , Orr Dunkelm & Nathan Keller **Linear Cryptanalysis of Reduced Round Serpent**, FSE 2002
xx B. Collard, F.-X. Standaert & J.-J. Quisquater **Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent** , Inscrypt 2007
xxi J. Kelsey, B. Schneier, and N. Ferguson, Yarrow-160: **Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator**, SAC '99 http://www.schneier.com/yarrow.html
xxii C. Kaufman (ed.), **Internet Key Exchange (IKEv2) Protocol**, IETF RFC 4306, http://tools.ietf.org/html/rfc4306
xxiii Nicolas T. Courtois and Josef Pieprzyk **Cryptanalysis of Block Ciphers with Overdefined Systems of Equations** http://eprint.iacr.org/2002/044
xxiv **MARS** , as cited above