Exploring Cipherspace:

Combining stream ciphers and block ciphers

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

November 2008

Table of Contents

Abstract:

This paper looks at the possibility of combining a block cipher and a stream cipher to get a strong hybrid cipher. It includes two specific proposals for combining AES-128 and RC4-128 to get a cipher that takes a 256-bit key and is significantly faster than AES-256, and arguably more secure. One is immune to algebraic attacks.

Overview

In this paper I want to look at the possibilities for using a stream cipher and a block cipher together to derive a cipher that is, in some senses, stronger than either. I think of this as “exploring cipherspace”; I'm meandering about looking for interesting unexplored corners.

The techniques discussed are applicable with any block cipher and any stream cipher. In principle, they might give good security in a high-performance design using some fast simple block cipher, or fewer rounds in a standard block cipher. They might also work with some unusual stream cipher. I will not look in detail at such possibilities, only sketch a few for purposes of discussion.

Instead, my proposed ciphers use AES [i] and RC4 [ii], comparing AES-256 versus a combination of AES-128 with RC4-128. This lets me concentrate on the combining techniques and the properties of the hybrid cipher. I simply assume that key length is known and that the underlying ciphers are reasonably strong and reasonably efficient.

Before getting to the actual proposals, I sketch out two thought experiment ciphers, intended to show that the stream cipher in a hybrid is hard to attack:

- One uses four-round DES; even a weak block cipher protects a stream cipher from attacks.
- The next uses a very simple operation instead of DES, to show that even that may help.

I then turn to the block cipher side of things. My actual proposals are two ciphers, one remarkably simple and the other fairly complex, that each combine RC4-128 and AES-128 to get a new cipher with a 256-bit key. I show that both are significantly faster than AES-256, and give arguments for their security.

Cipher speeds

It seems safe to say that stream ciphers are generally faster than block ciphers, though not hugely so. For any combination of ciphers, call the ratio of stream cipher speed to block cipher speed r. Generally, we will have $1 < r < 10$. Call the ratio of data encrypted to the amount of stream cipher output required t. Then the cost of combination cipher is approximately $1+1/rt$ times that of the block cipher alone.

For example, consider a 16-round block cipher. If rt is greater than 16, then using the combination technique gives higher encryption speed than adding a round. Since r is typically greater than 1 and we can adjust the design to give any t, this is easily achievable. I will try to demonstrate that, at least in some cases, using a combination also gives better security than adding rounds.

In comparing overheads, exact values are not critical for my arguments. I use benchmarks from Fornell at al.[iii]. They give cycles per byte encrypted on their system as 26.97 for RC4 and 206.19 for AES-128 in counter mode a ratio of 7.65 to 1. However, I choose to slightly underestimate the stream cipher advantage.

For RC4 and AES-128, I use $r = 7.5$ throughout the paper.

Stream cipher problem

All stream ciphers using a simple invertible operation as the combiner have a common theoretical vulnerability. Given some known or guessed plaintext, it is trivial for an adversary to recover some of the pseudorandom masking material.

For example, using P for plaintext, C for ciphertext, R for (pseudo)random data, and ^ for the most common combining function, bitwise exclusive OR, we get:

encryption: C = P^R
decryption: P = C^R

but this means the enemy (if he has P) can do:

recover R: R = C^P

Using another invertible operation, such as addition or even something more complex like IDEA multiplication, does not prevent an enemy from getting R; it only alters the details.

Getting R allows an active attacker to conduct a rewrite attack; he recovers the peseudorandom data and can then encrypt anything he chooses with it. If he can get his version delivered instead of the real one, this can be a disaster. Authentication at packet or message level completely prevents this attack, but stream ciphers (even one-time pads) themselves are inherently vulnerable to it.

A passive attacker can also get R. This is the first step in any attack on the pseudorandom generator in the cipher. Of course, against good ciphers such attacks are not expected to work. Also, in practice ciphers are re-keyed often enough that an attacker cannot get a large collection of material (R1, R2 ... Rn) enciphered with a single key, which both makes any break of a key less valuable to him and denies him the data required for some attacks.

Ritter [iv] suggests that using a more complex combiner, non-linear and with some internal state, is worthwhile because it prevents an enemy getting any pseudorandom output. I agree that this is an excellent suggestion, but I will not explore his proposed combiners here. Instead I look at using a very weak block cipher as the combiner. All I want to show is that using a non-linear combiner makes attacks harder.

Stream cipher + reduced round DES

To get an idea of what might be accomplished with even a weak block cipher, I look at using RC4 with a two-round or four-round version of DES as the block cipher.

Assume a 128-bit key. We load that into the stream cipher and use it to crank out a set of round keys for the weak DES. After each block encryption, we run the stream cipher some more and update some round keys. There is a choice here: update just one round key per block encrypted, update them all, or somewhere between? We need from one to four round keys, 48 to 192 bits of stream cipher output, per 64 bits actually encrypted; stream cipher overheads are from .75 to 3 times the cost of just using RC4.

DES is significantly slower than AES, but benchmarks vary. One [v] gives AES just under 2.5 times faster, which in my terms would give r ~= 18 for DES. Use 16, since it more convenient, and we get r~= 2 or r ~= 4 for 2-round and 4-round variants. Total overheads using four rounds and replacing all round keys every time are then about 7 times RC4. With 2-round DES and only one

round key replaced each time, just under 3 times RC4.

AES in counter mode is about 7.5 times slower than RC4, so those overheads are high for a stream cipher but not absurd. We might get a faster and stronger combination cipher by using 2-round AES or 4-round CAST-128. We could make this one stronger at almost no cost by having the stream cipher crank out two 64-bit whitening values and using 4-round DES-X rather than just DES. But I need an obviously weak cipher for the example, so I use 4-round DES.

To look at security of the combination, consider three hypothetical attackers. Tom can break DES easily by brute force. Dick has an algebraic crack; given one known plaintext and matching ciphertext, his formula instantly gives him the key. Harry hasn't actually broken DES, but he has a good brain, a substantial budget, and an encyclopedic knowledge of the relevant literature. Let us see how these three fare against our combination cipher.

Tom is completely defeated. Copacobana [vi]costs 9,000 Euros and breaks DES in a week. Tom has a big budget and good technology; perhaps he can break DES in a second. However, even two-round DES with independent round keys has 96 key bits; Tom needs 2^{40} seconds for that. If his hardware is very flexibly reprogramable, perhaps somewhat less; attacking 2-round DES might be 8 times faster per CPU and might allow 8 times as much parallelism. That gains him 6 bits, so he now needs 2^{34} seconds , still over 500 years. The four-round variant is out of his reach by astronomical amounts.

Dick is also completely defeated, if we replace all the round keys with new stream cipher output every round. With 64-bit DES blocks, one might write 64 rather complex boolean equations for output bits in terms of input bits and key bits. Plug in a known plaintext/ciphertext pair and you have 64 equations in 56 variables. Use two and you get 128 equations in 56 variables, and so on. We assume Dick has a solution for one of these systems.

However, with independent round keys, even two-round DES has 96 key bits. Dick might be the greatest algebraicist ever, with a one-page proof of the Reimann hypothesis and another for P < NP, but even then he cannot solve a system of 64 equations in 96 variables. Looking at multiple blocks gains him nothing if all of the key changes for every block; he just gets many insoluble systems, each with 64 equations and a different set of 96 variables.

If we replace only part of the round key array for each block, Dick might be able to solve the system. Consider four-round DES with one round key changed for each block; 192 round key bits and 48 new ones each for each block. Dick gets a system that is, in principle, soluble when:

$$64n >= 192 + 48(n-1)$$
$$n >= 9$$

n=9 gives him 576 equations in 576 variables. N=10 gives him 640 equations in 624 variables, and so on. In principle, such systems are soluble. However, they are a very different problem than DES, where he has only 56 variables to solve for. At the very least, Dick has a significant amount of work to do here. It is also possible that such a large system will be beyond his capabilities.

In general, however, if we mix in at least one block worth of stream cipher output per block encrypted, Dick is defeated. With 64 bits of stream cipher material used, the above equations for Dick's goal become:

$$64n >= 192 + \mathbf{64}(n-1)$$
$$0 >= 128$$

Achieving this is clearly impossile. Dick is defeated, no matter how many blocks he gets.

What about Harry? He reads all the papers, goes to all the conferences, and is on all the mailing lists. He has met all the players, and can quote all the important results from memory. Every paper of any importance is on his bookshelf or in his computer. Surely he can break four-round DES?

Yes, he almost certainly can. There is no shortage of relevant papers. Dunkelman et al. [vii] give a meet-in-the-middle attack on 6-round DES, and cite several earlier papers that also break reduced-round DES. Kelsey at al. [viii] give an attack on 16-round DES with independent round keys. Harry will no doubt find more papers than I did.

However, some attacks do not apply against independent round keys and others require many texts encrypted with the same key, which a hybrid cipher will never provide. Like Dick, Harry has some work to do before he breaks this. However, he is even more likely than Dick to break it eventually.

So the hybrid cipher defeats Tom and some variants defeat Dick. Two out of three ain't bad against amazingly strong hypothetical opponents. On the other hand, Harry almost certainly breaks it and Dick has a plausible, if expensive, attack on some variants.

However, breaking the block cipher leaves them looking at RC4. A combination cipher with a broken block cipher is no worse than a stream cipher that uses XOR. An attacker who breaks the block cipher can get the pseudorandom data whenever he knows or guesses the plaintext. So can an attacker who “breaks” the trivial XOR in a normal stream cipher. That puts him in a position to attack the pseudorandom number generator, but it does not give him an attack on the generator.

To summarise, then, even a weak block cipher like reduced-round DES may, when used in a hybrid system, completely defeat some opponents. Even when it does not, it makes attacks on the stream cipher significantly harder. A secure block cipher should make them impossible.

Other combiners

Almost any non-linear operation may be used as a combiner. Finding the ideal combiner, both efficient and secure, is a difficult problem which I will not address. Ritter proposes some, which I will not evaluate. However, finding a combiner that is both vastly more efficient than DES and much harder for the attacker than XOR is trivial, so I will give an example of that.

Suppose we use twice as much RC4 output, to get X and Y. Either take successive RC4 outputs or, probably better, run two copies of RC4, either with independent keys or one seeded from the other's output. Take everything in 32-bit blocks, call plaintext P and ciphertext C. Use a simple combiner, C = (P^X)+Y. This is reversible if you know X and Y, but an attacker who knows the plaintext and intercepts the ciphertext has no obvious way to infer X or Y. Use a more complex mixer such as IDEA multiplication, or throw in a third block Z, and it becomes even harder.

Similar things could be done with many other functions of comparable complexity. I do not know which functions are actually good. Possibly there are no functions that require little stream cipher data and are both fast and effective. However, anything along these lines is obviously better than XOR, giving a stream cipher that is harder to attack, though my example is only half the speed.

It seems possible that better variants exist; I would suggest that stream cipher designers should be looking for them. On the other hand, many stream ciphers already include a non-linear mixing step and it is possible that this trick adds nothing of value to them. In any case, my main concern is building a better block cipher, and I now turn to that.

Block cipher problem

All block ciphers share two common theoretical vulnerabilities, because they all encrypt multiple blocks with a single key. If an attacker can extract the key for one block, then he breaks other blocks with almost zero effort. Also, an attacker can collect many blocks encrypted with a single key, which may allow attacks that are impossible against a single block.

To illustrate, we return to Tom, Dick and Harry.

Tom needs only one block of known plaintext to try his brute force attack. Dick needs ceiling(keysize/blocksize) blocks for his algebra. If either of them succeeds, however, he gets everything ever encrypted with that key, often many more blocks.

Harry is a more complex case. With enough texts, he can try linear or differential cryptanalysis. Perhaps some smaller number enables some other attack; of course Harry has read the paper. With a really absurd number of blocks and enough storage, he might even build a codebook that breaks the cipher. Overall, we cannot be certain quite what Harry can do, but we do know re-keying often will stop some of it.

Of course we do not expect any of the three to succeed against well-designed real ciphers, and of course in sensible usage ciphers are re-keyed often enough to frustrate Harry, but in theory the vulnerabilities remain.

Mixing stream cipher output into the round keys completely eliminates these vulnerabilities, if we change the keys for every block. Since ideal round keys and ideal stream cipher output should both be effectively random, this seems a sensible thing to do.

If the key is different every time, then breaking one key gets Tom exactly one block of plaintext. However, he already had that block; it is required for his brute force attack. This makes his attack utterly worthless.

Dick has the same problem. If the key changes for every block, then doing the algebra to recover a key gains him exactly nothing. Also, if keysize is larger than blocksize, then he needs at least two blocks with the same key for his algebraic attack to work at all. If the keys change every time, then he cannot get those blocks.

To defeat Dick, we need not change all of the round keys. Suppose there are n bits per block and m bits of key that Dick wants to solve for, with m > n. He writes n equations in m+2n variables, then substitutes in a known plaintext/ciphertext pair to get n equations in m variables. Repeating this with a dfferent pair gives him a different set of n equations in the same m variables, so the overall system is 2n equations in m variables. Sufficient repetition gives him kn equations in m variables. If all the equations are linear, this becomes soluble when kn >= m. If they are non-linear, it is still soluble, at least in theory, for large enough k.

However, suppose we add n bits of stream cipher output for each block encrypted. One block of known plaintext still gives Dick n equations in m variables. The next block gives him n new equations, *but it also gives him n new variables*, the stream cipher output mixed in for that block. No matter how many blocks of known plaintext he gets, he can never have more equations than variables. He needs kn >= m + (k-1)*n and that never happens if m > n.

Even if the cipher were completely linear, he would never get a soluble system. Dick is utterly defeated, even if he has a complete and efficient algebraic crack for the block cipher itself. *Adding n bits of stream cipher material per n-bit block encrypted completely prevents any algebraic analysis of the block cipher.*

This remains true even if we mix the n bits into the cipher in some other way – in whitening data or in the auxiliary round keys which I describe later. As long as we use at least n bits and mixing is adequate, Dick is defeated.

Harry appears to be defeated as well. The only attacks he has against a single block are brute force, various sorts of algebra, or combinations of the two using algebra to constrain a search sufficiently that brute force can succeed. Those are blocked. All his other attacks require more than one block encrypted with the same key. They are all blocked by the key changes. The most general and powerful techniques -- linear and differential cryptanalysis -- require large samples of data encrypted with the same key. They are blocked as well.

However, variations on linear and differential cryptanalysis have been used against both block ciphers and stream ciphers, so there is no reason to suppose that a hybrid is necessarily completely immune. It stops Tom and Dick, but it may only give Harry a hard time.

Mixing methods

The question then arises how best to use stream cipher output to enhance a block cipher. Most of the rest of this paper deals with that, but first let us eliminate a few obviously flawed possibilities.

Ideally, one would change all the round keys for every block, completely defeating Tom and Dick and giving Harry a maximally difficult problem. However, this is generally too expensive.

Consider AES-128 with its 11 128-bit round keys. RC4 is roughly 7.5 times faster than AES-128, but even so using RC4 to update all round keys for every block almost triples the cipher overheads. We could use RC4 to generate a new 128-bit AES key for each block, but then we'd have to do AES key scheduling for each block, and that is not cheap either.

Just changing a few bits of the round keys for each new block is cheap, but it leaves the system open to incremental attacks, so it is not really effective. For example, suppose we only change one byte of the round key array after each block. This should be cheap enough; we need only one byte of stream cipher output for each 16-byte AES block so the overhead is only about 1+1/120 times that for AES alone.

However, if Tom or Dick breaks one block, he can then attack the next block (forward or back) knowing all but 8 bits of its round key. He can then iterate to recover all blocks encrypted with the same base key, everything up to the next rekeying and back to the previous one. This is not as bad as having those blocks use the identical key so he gets them for the cost of decrypting them, but it certainly is not ideal. An enemy who cracks one block gets many blocks, at the relatively moderate cost of 2^8 effort per block.

In their paper on the Yarrow random number generator, Schneier et al. talk about “catastrophic reseeding”, making a few big changes to the generator's state rather than many small ones, to prevent such iterative attacks. We clearly need something like that here, and there is an obvious minimum we can apply. For a cipher of block size n bits we want to change at least n bits at a time, preferably n bits per block encrypted to defeat Dick.

Keying the block cipher

There are choices to make in keying. Assume you have a stream cipher, a block cipher that needs a 128-bit key, and 256 key bits to work with. Do you use 128 bits for each cipher, or load all 256 into the stream cipher and let it generate the block cipher key? If the latter, should it generate a 128-bit key and let the block cipher key schedule create round keys, or should the stream cipher produce the round keys directly? If both ciphers are secure, and the key is good, then any of these approaches should be secure.

There are a few cases where using all of the key for the stream cipher is preferred. If you have a key that is small or weak, then it is better to use all the available entropy in one place. For example, if you have only 128-bit keys, or keys with only 100 bits of entropy, using those entire keys in the

stream cipher may give adequate security but splitting them up probably will not. If the block cipher has a simple key schedule, for example DES or IDEA, let the stream cipher generate round keys; they will then be more random, closer to independent keys. If the block cipher key schedule is expensive, as for Blowfish, using the stream cipher to produce round keys may be faster.

Other than those cases, either approach is fine.

Given a good large key, it is also possible to play it safe with a “belt and suspenders” approach. Give the block cipher its key and have it run its key schedule. Then load a different key into the stream cipher and XOR stream cipher output into the block cipher's round keys. This will be secure if *either* the key schedule or the stream cipher are. This is the preferred approach where applicable.

A simple hybrid cipher

We next look at a simple way to combine a stream cipher and block cipher; use stream cipher output as a pair of “whitening” values that are mixed into the plaintext before encryption and into the ciphertext after encryption.

In going from AES-128 to AES-256, the number of rounds increases from 10 to 14, a 40% performance hit. My question is whether, given availabilty of 256 key bits, one might use 128 for AES-128 and the other 128 for RC4, combining the two ciphers to get something that is in some way “better” than AES-256: in strength, in cost, or in the trade-off.

The first step is to mix RC4 output into the round keys at cipher setup time. Assuming RC4 gives effectively random output, algebraic attacks then become significantly harder. AES algebra no longer describes the key schedule. The algebraic attacker has to solve for all of the round keys, 176 bytes in AES-128, rather than just the 16-byte primary key. This requires both more known plaintexts and more algebra. The cost is a modest increase in cipher setup time.

Next, we add whitening. This is an established technique, used in DES-X [ix], RC6 [x], Blowfish [xi] and Twofish [xii]. Rogaway has an analysis [xiii]. Robshaw and Kaliski [xiv]show that DES-X is better than DES against both linear and differential cryptanalysis. In DES-X the whitening data is part of the key and in the other ciphers mentioned it is generated during key scheduling; in either case, it does not change thereafter. In a hybrid cipher, it can be changed as often as seems necessary or convenient.

Simply adding whitening also complicates algebraic attacks; the attacker has two more 16-byte chunks of data to solve for. Whitening that changes can make algebraic attacks impossible.

The simplest combining technique is to just apply the two ciphers in succession, in effect using the RC4 output as whitenting for the AES cipher. The overall algorithm is then:

$$\text{ciphertext} = X +_1 \text{AESencrypt}(\text{plaintext} +_2 Y)$$

Where X and Y are two 128-bit blocks of stream cipher output and $+_1$ and $+_2$ represent any two convenient combining operations. XOR is the obvious candidate; that is what Rivest used in DES-X [xv]. However, any invertible binary operation could be used. With AES, use 32-bit addition as the mixer; on most computers this is no more expensive than XOR and it is non-linear relative both to AES's algebra and to the XOR used to combine blocks in CBC-mode applications.

If RC4 is 7.5 times faster per byte than AES, this is significantly faster than AES-256. We are generating two blocks worth of stream cipher data for each block encrypted, adding 2/7.5 ~= 26% plus a tiny bit for the additions to the time cost. This is better than 40% for four extra rounds in

AES-256. Memory overheads are 2048 bits for RC4 state versus 512 for four extra AES round keys. Neither speed nor memory differences are large enough to be major factors in any choice. Code size and cipher setup time do increase; the increases appear reasonable, but they might be an issue in some applications.

We can do better. Change both whitening values in every round, using one block of stream cipher data and one block of block cipher output. This is about 1/7.5 or 13.33% slower than AES-128.

Calling the whiteners w0 and w1 and stream and block cipher outputs s and b. Mix them with something like:

```
w0 ^= s
w1 ^= b
sum = w0+w1
diff = w0-w1
w0 = sum
w1 = diff
```

This is fast and makes both outputs depend on all inputs. There may be a better way to mix them.

This uses one block of stream cipher output per block encrypted. Assuming the mixing is adequate, it meets the condition for completely defeating Dick and poses problems for Harry, as discussed above. Tom is completely out of the picture.

This simple combination, then, appears to defeat all known attacks short of brute forcing a 256-bit key. One possible weakness is obvious, though: the mixing function I suggest may not be adequate. This needs analysis and perhaps a better function.

This has overheads about 13% higher than AES-128. It can be strengthened further if increased overheads are acceptable. For 27%, we can use two blocks of stream cipher output per block encrypted; this allows us either to update all the round keys every 11 blocks or to mix one block of stream cipher material into the round key array for every block (ideally, non-linearly and into more than one round key). We could also update the round keys less often; for example changing them all every 64 blocks costs about 11/(7.5*64) times AES-128 speed, around 2%, so the cipher would be around 15% slower than AES-128 overall.

Summarising, then, we can derive several ciphers at speeds from 13% to 27% slower than AES-128 that are completely resistant to algebraic attack and apparently resistant to other attacks as well.

Supplemental round keys

For maximum confusion, one wants to alter the round keys, not just use whitening. However, any change to the round keys complicates analysis. In particular, it invalidates some of the existing analysis of the block cipher in question.

My solution is to have a set of supplemental round keys, in effect applying a mini-whitening operation in every round as well as whitening for the cipher as a whole. The principle is easily demonstrated using a Feistel cipher as an example.

In a Feistel cipher, using F for the round function and R_n for the round key in round n, even numbered rounds can be written as:

$$left_n = left_{n-1} \text{ ^ } F(right_{n-1}, R_n)$$
$$right_n = right_{n-1}$$

and odd-numbered rounds as the reverse. I propose adding supplemental round keys S_n and making the second line above:

$$right_n = right_{n-1} + S_n$$

and of course the reverse in the other rounds. The + represents 32-bit addition.

We do not need a strong mechanism here. Anything that mixes in quite a few more bits and is cheap, and preferably somewhat non-linear, is fine. Here the R and S arrays are the same size, so there's enough keying material used. Addition is cheap enough and is non-linear in relation to the XOR in the first line.

The first line is unchanged so (barring pathological cases like an enemy who controls some of your round keys) this cannot possibly weaken the cipher. It certainly increases strength against brute force and it adds some non-linearity which should help against other attacks as well.

In some ciphers, you may be able to do this quite cheaply. The CAST-128 [xvi] key schedule, for example, generates 64 bits per round key, but the actual cipher uses only 37 bits per round. The 27 unused bits per round could be used as S_n at very little cost.

AES is not a Feistel cipher, so we need a somewhat different mechanism. Each AES round has four operations. The first two give confusion and the last two give diffusion:

```
AddRoundKey: bitwise XOR of 128-bit state and 128-bit round key
SubBytes: run individual bytes through an 8*8 S-box
ShiftRows: cyclicly shift each row by a fixed amount
MixColumns: mix each column with a polynomial multiplication
```

I add one more operation so it is:

```
AddRoundKey: bitwise XOR of 128-bit state and 128-bit round key
SubBytes: run individual bytes through an 8*8 S-box
AddExtraKey: 32-bit additions of 128-bit state and extra 128-bit key
ShiftRows: cyclicly shift each row by a fixed amount
MixColumns: mix each column with a polynomial multiplication
```

This has the required properties. The S array is almost as large as the R array (n blocks rather than n+1 for an n-round cipher), so enough bits are used. Addition is cheap enough and is non-linear in relation to the algebra of AES.

This operation affects every bit of round output. Because it is done before the AES mixing steps, every byte of the extra round key affects at least four bytes of that round's output, possibly more because of carries, and all bytes of the following round.

There has been some work tending to show that the AES S-box is insufficiently non-linear [xvii]. If that is true, the small amount of extra non-linearity introduced by the additions here would help, but it would almost certainly not be enough. The addition also adds some extra diffusion because of carries; as for non-linearity, the amount is small, perhaps helpful but not enough overall.

What I propose below is a cipher that combines this method, whitening, and another method for fast cheap changes to the extra round keys. The block cipher, AES-128, gets its own 128-bit key and normal round keys are generated using the AES key schedule, to simplify analysis.

A more complex hybrid cipher

I now propose another cipher that combines AES-128 and RC4-128 in attempt to construct a cipher that takes a 256-bit key and is both faster than AES-256 and at least as secure.

Unlike the simpler cipher proposed above, this one never changes the round keys, which may simplify some analysis. This is a more complex cipher using auxiliary round keys. My main purpose here is to illustrate additional mechansims which are available. I do not consider these mechansims necessary; to me the simpler cipher is clearly a better design. However, it seems worth trying for completeness.

Two auxiliary data structures are used, two blocks of whitening data and a set of supplemental round keys, S. Both use data from the stream cipher, and the supplementary keys use block cipher output as well.

We want to change a great deal of the cipher state (ideally, all the round keys) for each block, and yet to keep overheads low. This appears impossible, so I compromise by using three operations – a fast cheap one that changes one supplementary round key per block encrypted, a medium-cost one that changes the whitening data every 36 blocks, and a heavy-duty expensive one that updates all the supplementary round keys every 200 blocks.

These numbers are of course somewhat arbitrary; one could use smaller ones to get more frequent changes or larger ones to get lower overheads. I chose these to make overall overheads for stream cipher usage under 1% of AES cost each, for both the medium and the heavy-duty operation. I also chose them so that one number was not a multiple of the other, since this complicates cryptanalysis marginally. Every 1800 blocks, however, both changes happen at once.

At setup time, we split the 256-bit key into two parts, use one to key AES-128 and one for RC4. Run RC4 a bit to bypass any weak initial output, then use it to initialise the whitening blocks and the supplementary round key array S. It would be a good idea to mix RC4 output into the main round key array as well, but I refrain here to keep the analysis simple.

After that initialisation, all operations on S are reversible so it can never become less random. We never overwrite any entry of S, only add or XOR new stuff into it.

Both whitening blocks are used in every round; one is mixed with the plaintext before encryption and the other with the ciphertext after encryption. The mixing operation is addition mod 2^{32}, chosen because both AES and CBC mode use XOR and we want something different. The overheads for this are small; 8 32-bit additions per round.

Using a block of stream cipher output per block encrypted is too expensive for a fast cheap operation; it adds about one sixth to cipher cost. Instead I mix the block cipher output into S, changing one element of S per block encrypted. This is cheap and block cipher output is suitably random for this usage. Note that this technique could be applied even to a block cipher used alone, with no stream cipher in the picture, as I have suggested elsewhere [xviii].

The only disadvantage is that an enemy can know the data. This may not be important, since he cannot know what it is being mixed with. However, I add a cheap complication so he cannot know exactly how it is mixed either.

Every 32 blocks, we get 32 bits of stream cipher output. We get small amounts often to ensure that when one of the less frequent operations fetches stream cipher data it will not get sequential data on successive calls; this small fetch will always be in between to break up the sequence. This is almost

certainly not necessary, but it is cheap and can do no harm so it seems worthwhile.

For each operation mixing block cipher output into the S array, we use one bit of stream cipher data to choose between bitwise XOR and 32-bit addition as the mixing operation. In the long term, this is enough to make the operation non-linear and prevent an enemy from knowing how the mixing went. In the short term, it isn't, but it seems as good as we can get in a fast cheap operation. The stream cipher data used is only one bit per block encrypted, so t = 128 and stream cipher overhead is about 1/(7.5*128) or 1/1000 of AES cost.

My intermediate-level operation is to update the two whitening values every 36 blocks. This uses one block of stream cipher data per 18 blocks encrypted, so stream cipher overhead is 1/135 of AES cost.

The heavy-duty operation updates all ten supplementary round keys every 200 blocks. This uses one block of stream cipher data per 20 blocks encrypted (only 10 keys in S, not 11), so stream cipher overhead is about 1/150 of AES cost.

Total stream cipher overheads are then 1/1000 + 1/135 + 1/150 times AES cost, well under 2%. The overheads added to the block encryption itself are:

two block additions (4 32-bit additions each) per block to apply the whitening
one block addition per round for the supplemental round keys

and the overheads for the three data-updating operations:

a shift, a branch and one block addition or XOR per block, to update S from output
two block writes every 36 blocks, updating whitening data
10 block XORs every 200 blocks to update the supplementary keys

Ignoring the shift and branch, and considering all block operations equal, this averages out to about 13.1 block operations per block encrypted. AES has 4 steps per round, so AES-128 has 40 steps per block encrypted. This cipher cannot possibly be worse than 35% slower than AES-128, 2% for the stream cipher and 33% for extra operations in the block cipher. Allowing for the fact that some AES steps are more complex than the simple block operations, the 33% is likely an over-estimate.

At any rate, this cipher should be signficantly faster than AES-256, which is 40% slower than AES-128 because of the extra rounds. Strength against brute force is fundamentally the same, 256 bits for either AES-256 or this cipher.

Other security considerations are harder to assess. This cipher is an odd design. Many of the standard attacks do not apply, so arguably it is extremely secure. On the other hand, much of the standard analysis does not apply either, so it is difficult to have any confidence in it.

This cipher cannot be weaker than AES-128. All of AES-128 is still there; all we add is whitening at the block level and the round level. It looks as though it ought to be stronger because of all the whitening, but it hard to be certain of that, let alone quantify it.

AES-256 has 15 round keys; this has 11 primary and 10 extra round keys, plus the two whitening blocks. This cipher mixes a total of 23 blocks of keying material into each block encrypted, versus 15 blocks for AES-256, but the mixing functions in this cipher are not as good. It is not clear which is better, or if the difference is unimportant.

No-one is worried about Tom here. He is not going to brute force a 128-bit key, he most definitely is not going to brute force a 256-bit key, and he is not even going to consider a brute force search for round keys.

Dick is perhaps more of a worry. AES has a lovely algebraic structure [xix]and it seems conceivable that some clever attack exploiting that struture can be found. Indeed, Courtois and Pieprzyk [xx] claim to have already found principles that could lead to such an attack; their work is widely acknowledged as interesting, but not all their claims are generally accepted.

Someone finding an actual attack appears rather unlikely, since quite a few very good people – all the teams for the other AES candidates, for starters – have looked hard and not come up with anything, but it is certainly not inconceivable.

This cipher complicates any such attack immensely. Not only are there quite a few more variables to solve for -- S and the whitening blocks as well as the round keys -- but they are changing all the time and the operations that mix them in are not easily described in AES algebra.

Suppose Dick does somehow manage to break both ciphers. Against AES-256, he then gets all blocks up to the next key change and back to the previous one essentially free, for the cost of decrypting them.

Against this cipher, he cannot even get adjacent blocks that cheaply; he needs two test decryptions per block to see if XOR or addition was used for the intervening change of S. Of course this is only a minor annoyance to him, but it does double his costs. After 36 blocks he encounters a larger annoyance; both whitening values change. Then after 200 blocks, all the S-values change. At 1800 blocks he hits a point where both whitening and S-values change simultaneously. If these changes stop him, then they limit the damage his attack does; he gets only 36, 200 or 1800 blocks per crack, rather than all blocks up to key changes. Even if they fail to stop him, they should at least increase his costs.

Note, however, that unlike the earlier simpler cipher this one is not guarnteed to stop Dick. There are changes that may stop him, but not enough change per round to definitely rule out his attacks.

As usual, Harry is a more complex case, and of course he is the important case. We certainly are not worried about brute force here, and algebraic attacks appear to be an improbable risk. However, clever attackers who have substantial resources and know the literature are a very plausible threat.

We can say with some confidence that attacking the stream cipher will be extremely difficult; even four-round DES makes that significantly harder, so AES-128 should make it nearly impossible. We could use a really weak stream cipher and not expect an attacker to break it, since he cannot see any of its output. We do not expect him to be able to even start an attack on RC4 here unless he first breaks the block cipher.

Add the very plausible assumption that RC4 is a good stream cipher, that the data it generates looks very close to random to anyone who does not know the key. The whitening data and the extra round keys must then be effectively random from Harry's point of view. They will therefore be maximally annoying to him, complicating his analysis as much as any data used that way possibly could.

The data is used in apparently sensible ways. Whitening is a well-known technique, widely used and well analysed. I extend it by using the extra round keys, effectively doing whitening for every round as well as for the whole cipher. Of course this extension needs analysis, but it is at least superficially plausible.

This cipher defeats any attack that needs more than one block encrypted with the same key. This includes the usual forms of both differential and linear cryptanalysis, both of which need large numbers of plaintext/cipertext pairs encrypted with the same key.

This cipher defeats the best known attacks on AES. In one of the Twofish team's papers [xxi] there is a table giving eight known attacks that break reduced-round Rijndael (now AES) with 6 to 9 rounds with various amounts of effort. The lowest amount of chosen plaintext required is 2^{32} blocks, encrypted with one key. Against the cipher proposed here, an attacker cannot get even two blocks with exactly the same key and cannot get even 2^8 without a complete change to the extra round keys.

Of the attacks in that paper, the one that breaks the most rounds is: “a related-key attack on 9 rounds of Rijndael with 256-bit keys that uses 2^{77} plaintexts under 256 related keys, and requires 2^{224} steps”. If we use RC4 output for the primary round keys, then that attack fails. Of course there might be a related key attack on RC4, though none is known. If we are worried about that, we can first do AES key scheduling and then XOR in RC4 output; then an attacker needs related-key attacks on both to get anywhere.

The actual proposed cipher uses the AES round keys, so the described related-key attack might apply. However, that attack needs 2^{77} plaintexts each for 256 related keys; the proposed cipher changes the whitening every 36 blocks and the entire S array every 200, so nothing like 2^{77} texts encrypted the same way can be collected. Also, the introduction of whitening and the extra round keys should complicate the attack considerably. The attack needs 2^{224} effort against 9-round Rijndael-256; it seems realistic to hope that the effort against the combination cipher, with 10-round AES-128 plus complications, would be high.

Conclusion

I conclude that the technique of using a stream cipher and a block cipher together has the potential to yield reasonably efficient ciphers which are resistant to all known attacks.

There are a whole range of design choices, once the basic notion of combining stream ciphers and block ciphers is accepted. I almost certainly have not found all the interesting ones.

i National Institute Of Standards and Technology AES links; http://csrc.nist.gov/groups/ST/toolkit/block_ciphers.html
ii RSA Laboratories RC4 page: http://www.rsa.com/rsalabs/node.asp?id=2250
iii N. Fournel1 , M. Minier2 , and S. Ubeda, "Survey and Benchmark of Stream Ciphers for Wireless Sensor Networks " http://wistp2007.xlim.fr/Slides2007/Day2/WISTP2007-SmallDevices-p3.pdf
iv Terry Ritter's web site, http://www.ciphersbyritter.com/GLOSSARY.HTM#StreamCipher
v Open SSL cipher benchmarks, http://www.madboa.com/geek/openssl/#benchmark
vi Copacobana FPGA-based cipher cracker (http://www.copacobana.org/)
vii Orr Dunkelman, Gautham Sekar, and Bart Preneel "Improved Meet-in-the-Middle Attacks on Reduced-Round DES", LNCS, http://www.springerlink.com/content/h04jr5868154/?p=9e4a5f9304be42c4b20e135848acf745&pi=0
viii John Kelsey, Bruce Schneier & David Wagner "Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES"
ix RSA Laboratories DESX page: http://www.rsa.com/rsalabs/node.asp?id=2232
x RSA Laboratories RC6 page: http://www.rsa.com/rsalabs/node.asp?id=2512
xi Bruce Schneier's Blowfish page: http://www.schneier.com/blowfish.html
xii Bruce Schneier's Twofish page: http://www.schneier.com/twofish.html
xiii Phillip Rogaway "The Security of DESX", *www.cs.ucdavis.edu/~rogaway/papers/cryptobytes.ps*
xiv Matt Robshaw & Bob Kaliski "Multiple Encryption: weighing security and performance", Doctor Dobb's Journal, January 1996
xv RSA Laborarories web page "What is DESX?", http://www.rsa.com/rsalabs/node.asp?id=2232
xvi RFC 2144 "The CAST-128 Encryption Algorithm" http://www.ietf.org/rfc/rfc2144.txt
xvii Joanne Fuller & William Millan "On Linear Redundancy in the AES S-Box", http://eprint.iacr.org/2002/111
xviii Sandy Harris "Is this a new mode?", post to sci.crypt newsgroup, Oct 1998 http://groups.google.com/group/sci.crypt/browse_frm/thread/355f901822f00e11?ie=UTF-8&oe=utf-8
xix Niels Ferguson, Richard Schroeppel, and Doug Whiting "A simple algebraic representation of Rijndael", SAC 2001, LNCS #2259, pp. 103–111, Springer Verlag, 2001. http://www.macfergus.com/pub/rdalgeq.html
xx Nicolas T. Courtois and Josef Pieprzyk "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations " http://eprint.iacr.org/2002/044
xxi Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Mike Stay, David Wagner, and Doug Whiting "Improved Cryptanalysis of Rijndael", http://www.schneier.com/paper-rijndael.html