Extended TEA Algorithms

Tom St Denis
April 20th 1999

Abstract. This paper presents some natural manners to use TEA [1] and XTEA [2] in a variety of designs while improving security and keeping with the original design criteria.

Introduction

The TEA family of ciphers are relatively strong. TEA was cryptanalysis and found to be quite secure. Except for the weak key schedule the algorithm is quite good. The authors fixed some of the simpler errors in the key schedule and designed XTEA. This paper is intended to demonstrate how to make XTEA more secure, and apply it in a variety of different designs.

All the pseudo-code will be C based, this is because it's intended for drop-in use. The code is relatively simple to read and understand however.

XTEA-1

The only obvious problem with XTEA [2] is that the bits in keys will always effect the same bits in each round. This hasn't been exploited yet, but that doesn't mean it can't be. This first proposal is a 64 bit block cipher, which features 128 scheduled keys. The keys are scheduled dynamically at runtime, and require no memory. The scheduled keys are derived from performing plaintext-dependant cyclic bit rotations on the keys. Assuming that all five bits used in the rotation count have a probability of 1/2 and each key word has a probability of 1/4, each scheduled key would have a probability of 1/128.

The C code for encoding is

```
void xtea1(unsigned long key[4], unsigned long plain[2])
{
	unsigned long y, z, sum, r;
```

```
	/* load and pre-white the registers */
	sum = 0;
	y = plain[0] + key[0];
	z = plain[1] + key[1];

	/* Round functions */
	for (r = 0; r < ITERATIONS; r++) {
		y += ((z << 4) ^ (z >> 5)) + (z ^ sum) + rol(key[sum & 3], z);
		sum += 0x9E3779B9;
		z += ((y << 4) ^ (y >> 5)) + (y ^ sum) + rol(key[(sum >> 11) & 3], y);
	}

	/* post-white and store registers */
	plain[0] = y ^ key[2];
	plain[1] = z ^ key[3];
}
```

Where plain[] is the plaintext, and also the destination for this function. ITERATIONS is the number of iterations you wish to perform. Currently anything between 16 and 32 (32 to 64 rounds) is considered acceptable, but the authors of TEA suggest 32. The function 'rol' is used to perform cyclic bit rotation of the key, using the lower 5 bits of the register for the count. The decryption routine is the same, except you use subtraction, and the rounds are in the opposite order.

This algorithm requires only one rotation per round, which is quite effective and fast on most modern processors. It features the same mixer/diffusion (which has not been publicly broken) and the a similar (yet expanded) key schedule.

The post-white can also be extended to use a rotation, with similar benefits as in the round function. The rotations can be extended to use '(y >> 27) ^ y' or '(z >> 27) ^ z' such that both the upper and lower five bits are used to calculate the order of the scheduled keys. This means that statistically more errors in the key schedule will be caused if the wrong values (key or plaintext) are used.

XTEA-2

You can expand the previous to use a 128 bit block. Which may prove to be more usefull. It would require slightly more iterations (32 to 64), but would encode more information faster. The general C code would resemble

```
void xtea2(unsigned long key[4], unsigned long plain[4])
{
	unsigned long a, b, c, d, sum, r, t;

	/* load and pre-white the registers */
	sum = 0;
	a = plain[0];
	b = plain[1] + key[0];
```

```
        c = plain[2];
        d = plain[3] + key[1];

        /* Round functions */
        for (r = 0; r < ITERATIONS; r++) {
                a += ((b << 4) ^ (b >> 5)) + (d ^ sum) + rol(key[sum & 3], b);
                sum += 0x9E3779B9;
                c += ((d << 4) ^ (d >> 5)) + (b ^ sum) + rol(key[(sum >> 11) & 3], d);

                /* rotate plaintext registers */
                t = a; a = b; b = c; c = d; d = t;
        }

        /* store and post-white the registers */
        plain[0] = a ^ key[2];
        plain[1] = b;
        plain[2] = c ^ key[3];
        plain[3] = d;
}
```

The major benefits of this algorithm is that you can encode a larger block in a similar manner. It uses the same basic operations as XTEA-1, but requires more iterations. Technically speaking it requires exactly double the number, about 32 to 64 iterations (64 to 128 rounds). Perhaps 48 iterations (96 rounds) would be a good compromise between speed and security.

The decryption code would be a little more complicated to write, but still quite simple. It involves unrotating the key, then performing the rounds in reverse order. The post-white like XTEA-1 could involve rotation of the key material.

XTEA-3

Another natural extension (which would be slower) would be to use a 256 bit key. It would probably require the upper limit of rounds (32 iterations, 64 rounds), but provide greater difficulty against brute force searches.

```
void xtea3(unsigned long key[8], unsigned long plain[4])
{
        unsigned long a, b, c, d, sum, r, t;

        /* Load and pre-white the registers */
        sum = 0;
        a = plain[0] + key[0];
        b = plain[1] + key[1];
        c = plain[2] + key[2]
        d = plain[3] + key[3];

        /* Round functions */
        for (r = 0; r < ITERATIONS; r++) {
                a +=    ((b << 4) + rol(key[(sum % 4) + 4], b)) ^ (d + sum) ^
```

```
                ((b >> 5) + rol(key[sum % 4], b >> 27);

        sum +=  0x9E3779B9;

        c +=    ((d << 4) + rol(key[((sum >> 11) % 4) + 4], d)) ^ (b + sum) ^
                ((d >> 5) + rol(key[(sum >> 11) % 4], d >> 27);

        /* rotate registers */
        t = a; a = b; b = c; c = d; d = t;
    }

    /* Store and post-white the registers */
    plain[0] = a ^ key[4];
    plain[1] = b ^ key[5];
    plain[2] = c ^ key[6];
    plain[3] = d ^ key[7];
}
```

This design features a 128 bit block, with a 256 bit key. It uses the top five and bottom five bits of the plaintext register for the rotation count, because statistically speaking these are more likely to be susceptible to change. It is quite a bit slower, although as compared to [3] could be considered acceptable. It would still require the minimum of 32 iterations (64 rounds), however 48 iterations (96 rounds) may be acceptable tradeoff between speed and security.

This design is similar to TEA, but mixes a 256 bit key instead of a 128 bit key. Given the equal probability of the keys being used, and the equal probability of each bit in the register, there would be 8 * 32 or 256 scheduled keys, each with a probability of 1/256.

Conclusion

All three algorithms which have been proposed in this paper, are natural extensions of the original TEA [1] and X-TEA [2] algorithm. They feature better key scheduling and larger block sizes.

The use of dynamic plaintext dependant key scheduling means that there is no preset order for the use of the scheduled keys, and that they require no memory. This is quite a useful property as detecting which scheduled keys were used is most likely a difficult task. The key schedule is most likely more resistant to differential analysis since the bits in the key can effect any 1/32 possible other bits. The use of non-linear algebra (mixing addition with binary exclusive or) is considered effective against linear analysis. There are no known weaknesses in the use of mixed algebraic operations either.

The first design (XTEA-1) is the fastest, and given enough rounds (about 32 to 64) is probably secure. XTEA-2 presents a manner of extending the algorithm to larger blocks, it is no more secure then XTEA-1 however. XTEA-3 presents a manner of using

a larger key and larger block size. This design is quite a bit slower, but probably more secure.

Public scrutiny is required to validate the difficulty of 'cracking' this algorithm, however since it's based on the original TEA cipher, with corrections to the only known flaws, it is believed to be rather strong. Also the number of rounds (specifically the minimum) need to be determined. Presently the minimum of 32 rounds per 64 bit block is not too much (as the operations performed are rather simple), but the possibility of maintaining security with fewer rounds seems tempting.

Personally (and objectively) I believe that XTEA-3 is not best of the three presented in this paper. It presents a good design, but needs further work. The first problem is that only the top or the bottom five bits are used for the rotation amounts. Like XTEA-1 (and XTEA-2) the top five and bottom five bits should be used in each rotation. Another problem is that the key is divided into two four element subsets, and each part of the equation uses only one of the subsets. This should be extended to use any one of the eight subkeys in any part of the equation. If these additions could be made, the algorithm would be more practical, but given that it really is moving away from the TEA design criteria, I have not explored it any further.

Table of Specifications

Name	Min Rounds.	Max Rounds.	Block Size	Key Size
XTEA-1	32	64	64 bits	128 bits
XTEA-2	64	128	128 bits	128 bits
XTEA-3	64	128	128 bits	256 bits

References

[1] TEA a Tiny Encryption Algorithm, David J. Wheeler & Roger M. Needham

[2] TEA Extensions, Roger M. Needham & David J. Wheeler

[3] The RC6 Block Cipher, An AES proposal, Ronald L.Rivest, M.J.B Robshaw, R.Sidney, Y.L Yin.