The Design of the ICE Encryption Algorithm

Matthew Kwan

mkwan@cs.mu.oz.au

Abstract. This paper describes the design and implementation of the ICE cryptosystem, a 64-bit Feistel block cipher. It describes the design process, with the various aims and tradeoffs involved. It also introduces the concept of keyed permutation to improve resistance to differential and linear cryptanalysis, and the use of an extensible key schedule to achieve an explict tradeoff between speed and security.

1 Introduction

The Data Encryption Standard (DES) [8] has been widely used as an international standard since its introduction in 1977. However, in the years since its release, a number of vulnerabilities have come to light.

These include susceptibility to differential cryptanalysis [2], susceptibility to linear cryptanalysis [7], a key/plaintext complementation weakness [4], four weak and twelve semi-weak keys [4], a fixed 56-bit key size, inefficient software performance, and an absence of public design criteria.

While triple-DES [10] provides a larger key size at 112 bits, this is at the expense of a factor of three in encryption speed.

ICE, which stands for *Information Concealment Engine*, was designed to address these issues, while maintaining a compatible interface with DES. This is to allow it to act as a substitute in existing applications.

In addition to the standard ICE algorithm, other variants are described. Thin-ICE is a faster, less secure version, while there are also open-ended variants ICE-n which trade off greater key size for reduced encryption speed.

2 The Structure

ICE is a standard Feistel block cipher, with a structure similar to DES.

It takes a 64-bit plaintext, which is split into two 32-bit halves. In each round of the algorithm the right half and a 60-bit subkey are fed into the function F. The output of F is XORed with the left half, then the halves are swapped. This repeated for all but the final round, where the final swap is left out, as is illustrated in figure 1. At the end of the rounds, the halves are concatenated to form the ciphertext.

Decryption follows the same procedure, except that the subkeys are used in reverse order.

A Feistel structure was chosen for a number of reasons. To begin with, it was guaranteed to carry out one-to-one mappings between plaintext and ciphertext,

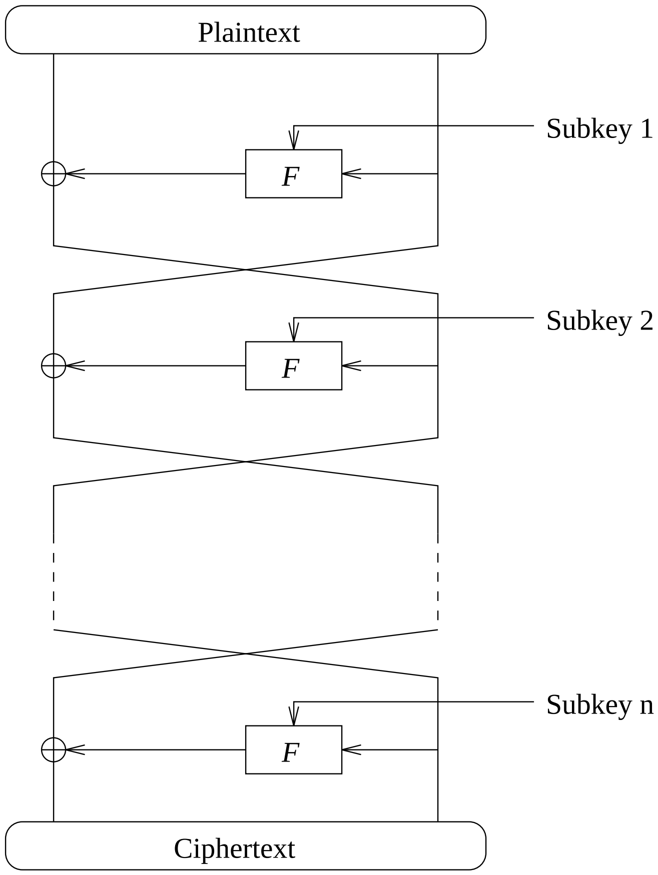

Fig. 1. The structure of n-round ICE

which is necessary for a cipher to be decryptable. This enabled the designer to concentrate on the design of the F function and key schedule, secure in the knowledge that a valid cipher would be produced.

Secondly, Feistel ciphers have been publicly cryptanalysed for more than two decades, and no systematic weakness has been uncovered. In addition, the techniques that have been used to analyse existing Feistel ciphers are generally applicable to new ones. This simplifies the design task, since the designer is not forced to invent as many new forms of cryptanalysis when evaluating a design.

And finally, Feistel ciphers are reasonably fast and simple to implement in software. Speed and simplicity were two important design aims for ICE.

3 The *F* Function

Notation: In this paper, bits will be numbered from right to left, starting at bit zero. So, for example, the rightmost bit of a plaintext half is P_0, while the leftmost bit is P_{31}.

The ICE F function is similar in structure to the one used in DES, with the exception of keyed permutation (described below). The function as a whole is illustrated in figure 2.

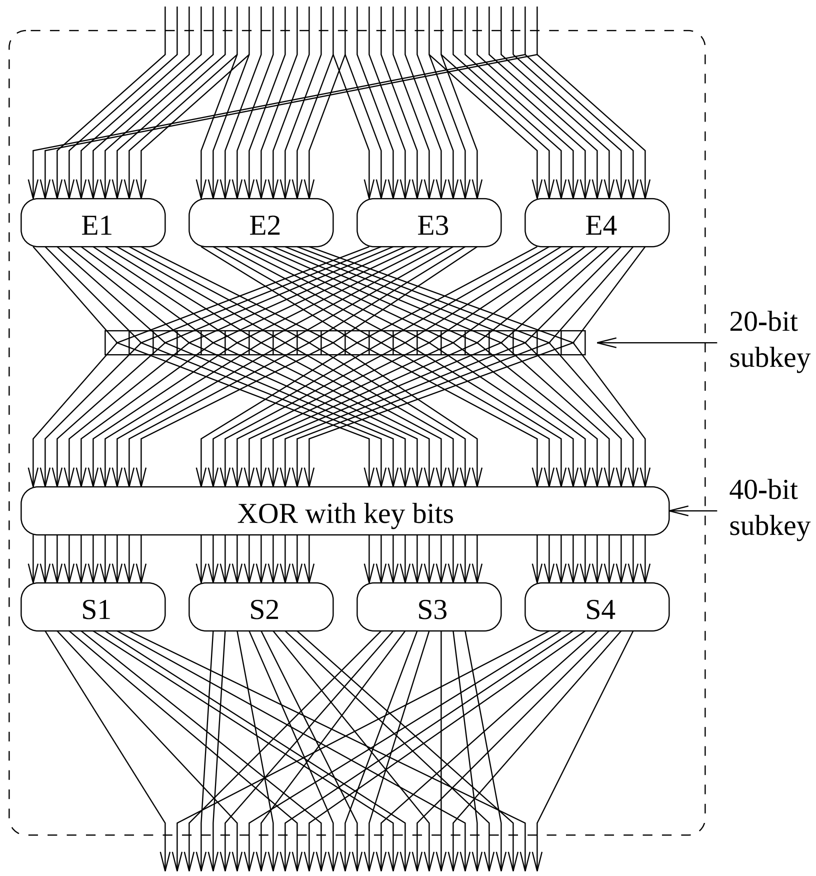

Fig. 2. The ICE F function

3.1 The Expansion Function E

The 32-bit plaintext half is expanded to four 10-bit values, E1, E2, E3, E4, in the following manner.

$$
\begin{aligned}
\texttt{E1} &= P_1 P_0 P_{31} P_{30} P_{29} P_{28} P_{27} P_{26} P_{25} P_{24} \\
\texttt{E2} &= P_{25} P_{24} P_{23} P_{22} P_{21} P_{20} P_{19} P_{18} P_{17} P_{16} \\
\texttt{E3} &= P_{17} P_{16} P_{15} P_{14} P_{13} P_{12} P_{11} P_{10} P_9 P_8 \\
\texttt{E4} &= P_9 P_8 P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0
\end{aligned}
$$

This expansion function was chosen because four 10-bit values were needed for the S-boxes, and it was reasonably fast to implement in software.

3.2 Keyed Permutation

After expansion, keyed permutation is used. The permutation subkey is 20 bits long, and is used to swap bits between E1 and E3, and between E2 and E4. For example, if bit 19 of the subkey is set, bit 9 of E1 and E3 will be swapped. If bit 0 of the subkey is set, bit 0 of E2 and E4 will be swapped. The cryptographic properties of keyed permutation are described in a later section.

The values E1, E2, E3, and E4, after being permuted, are XORed with 40 bits of subkey, then used as input to the four S-boxes, S1, S2, S3, and S4. Each S-box takes a 10-bit input and produces an 8-bit output. The S-boxes are described in more detail later.

3.3 The Permutation Function P

The four 8-bit S-box outputs are combined via a P-box into a 32-bit value, which becomes the output of the F function. The P-box, which is specified in table 1, was designed to maximize diffusion from each S-box, and to ensure that bits which are separated by 16 places never come from the same S-box, nor from S-boxes separated by two places (eg S1 and S3). Given that these criteria were met, the P-box was also designed to be as regular as possible.

Output bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Source	$S1_7$	$S4_7$	$S3_7$	$S2_7$	$S2_6$	$S3_6$	$S1_6$	$S4_6$	$S3_5$	$S2_5$	$S4_5$	$S1_5$	$S4_4$	$S1_4$	$S2_4$	$S3_4$
Output bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Source	$S2_3$	$S3_3$	$S4_3$	$S1_3$	$S1_2$	$S4_2$	$S2_2$	$S3_2$	$S4_1$	$S1_1$	$S3_1$	$S2_1$	$S3_0$	$S2_0$	$S1_0$	$S4_0$

Table 1. The permutation function P

The 32-bit output is thought of as eight 4-bit blocks, and diffusion is achieved by ensuring that each bit from a particular S-box is permuted to a different 4-bit block in the output. This means that the output of one S-box is guaranteed to affect the inputs to all the four E-boxes in the next round. Whether it affects the inputs to all the S-boxes depends to some extent on the keyed permutation in that round.

Although each S-box output bit goes to a different 4-bit block, there are four positions within each block to choose from. For an even spread, it was decided that each position would be used twice (since there are eight bits coming out of each S-box).

Because of the E-boxes, certain input bits to the F function (bits 0, 1, 8, 9, 16, 17, 24, 25) end up affecting two S-boxes, while the other bits only affect one. The P-box was designed to ensure that only one bit from each S-box will exclusively affect that same S-box in the next round, assuming that the bit isn't redirected by the keyed permutation. However, in the worst case, keyed permutation can result in up to three bits from an S-box exclusively affecting the same S-box in the next round.

4 The S-boxes

The S-boxes in ICE are similar in structure to those used in LOKI [3] in their use of Galois Field exponentiation.

Each S-box takes a 10-bit input X. Bits X_9 and X_0 are concatenated to form the row selector R. Bits $X_8..X_1$ are concatenated to form the 8-bit column selector C. For each row R, there is an XOR offset value O_R, and a Galois Field prime (irreducible polynomial) P_R.

The 8-bit output of an S-box for an input X is given by $(C \oplus O_R)^7 \ mod \ P_R$, under 8-bit Galois Field arithmetic. The exponent 7 was chosen because it is a one-to-one function, and it produces a reasonably flat XOR profile, useful for resistance to differential cryptanalysis. Regardless of the prime used as the modulus, no input difference produces an output difference with a probability greater than $\frac{6}{256}$.

The XOR offsets for each row in each S-box are given in table 2, while the prime numbers are specified in table 3.

S-box	O_0	O_1	O_2	O_3
S1	83	85	9*b*	*cd*
S2	*cc*	*a*7	*ad*	41
S3	4*b*	2*e*	*d*4	33
S4	*ea*	*cd*	2*e*	04

Table 2. The S-box XOR offsets (in hexadecimal)

S-box	P_0	P_1	P_2	P_3
S1	333	313	505	369
S2	379	375	319	391
S3	361	445	451	397
S4	397	425	395	505

Table 3. The S-box Galois Field primes

The choice of these values is described in detail in a later part of this paper. The way the S-boxes are specified, with 16 offsets and 16 primes, was chosen so that the boxes were parameterised. This enabled software to automate the generation and evaluation of millions upon millions of different possible S-boxes.

5 The Key Schedule

ICE has been designed with an extensible key schedule to permit a tradeoff between speed and security.

The standard ICE algorithm takes a 64-bit key and uses 16 subkeys in 16 rounds. There is a fast variant Thin-ICE which uses 8 rounds with a 64-bit key, and there are open-ended variants ICE-n which use $16n$ rounds and $64n$-bit keys. For example ICE-2 uses 32 rounds and a 128-bit key.

5.1 Design criteria

When the key schedule was designed, a number of criteria were used.

- There must be no weak keys. In other words, for an n-round schedule there must be no two keys $K1$ and $K2$ such that the all the subkeys $SK(K1, i) = SK(K2, n - i + 1)$ for $i = 1..n$
- Only a single key complementation weakness, ignoring keyed permutation. In other words, assuming that bits aren't being permuted in the F function, the only values of A, B, C s.t. $ICE(P, K) = ICE(P \oplus A, K \oplus B) \oplus C$ is when A, B, and C have all bits set. This is largely unnecessary, since keyed permutation makes it impossible to exploit complementation weaknesses, but it can't hurt.
- Each subkey bit should only be dependent on only one key bit. This simplifies the proof [5] that the above two conditions are satisfied.
- No meet-in-the-middle attacks. This means that, for any round N, all key bits must be used either in the preceeding rounds, or all must be used in the following rounds.
- Since the F function makes use of 60 key bits per round, each key bit must be used 15 times in the ICE and ICE-n ciphers. Thinking of the 60 key bits as being divided into 15 4-bit blocks, each key bit must use a different 4-bit

block each time it is used. For Thin-ICE, with only eight rounds, every key bit must be used 7 or 8 times, and spread out such that, if the 60 bits are partitioned into three 20-bit blocks, each key bit is used in a 20-bit block 2 or 3 times.
- Immunity to related-key cryptanalysis [1]. This means some sort of irregularity in the key schedule.
- The key scheduling algorithm must be simple to implement in software, yet not too fast, so as to hinder exhaustive key searches.

5.2 Key schedule specification

The Thin-ICE key schedule is simply the first 8 rounds of the standard ICE key schedule. The ICE-n key schedules build on the ICE key schedule using the following algorithm.

```
Start with an empty key schedule.
for i = 1 .. n
  Take the next 64 bits of the key and use them to generate
      a 16-round ICE key schedule.
  Take the schedule so far, split it at the half-way point,
      and insert the new 16-round schedule.
end
```

This ensures that there is no simple meet-in-the-middle attack, and that if ICE has no weak keys, then neither does ICE-n. However, this structure is still susceptible to more sophisticated meet-in-the-middle attacks [11].

In order to satisfy the condition that there be no weak keys, it was firstly necessary to prevent an all-zero key from producing zero subkeys in all rounds. The condition that each subkey bit be dependent on only one key bit means that this can only be achieved by inverting certain bits during the key schedule.

In each round 60 key bits are used. These are typically stored in three 20-bit values, `SK1`, `SK2`, and `SK3`.

`SK1` is the value XORed with the inputs to S1 and S2.
`SK2` is the value XORed with the inputs to S3 and S4.
`SK3` is the value used for key permutation.

The key is first converted into four 16-bit blocks, `KB[0 .. 3]`. These blocks are used, along with the key rotations KR shown in table 4, to derive the subkeys in each round using the algorithm as follows.

`KB[0]` $= K_{63}..K_{48}$
`KB[1]` $= K_{47}..K_{32}$
`KB[2]` $= K_{31}..K_{16}$
`KB[3]` $= K_{15}..K_{0}$

```
for each round n = 1 .. 16
  for SK = SK1, SK2, SK3 in turn, 5 times each
    for i = 0 .. 3
      Set B to bit 0 of KB[(i + KR[n]) mod 4].
      Shift SK left one bit.
      Set bit 0 of SK to B.
      Shift KB[(i + KR[n]) mod 4] right one bit.
      Set bit 19 of KB[(i + KR[n]) mod 4] to the inverse of B.
    end
  end
end
```

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Rotation	0	1	2	3	2	1	3	0	1	3	2	0	3	1	0	2

Table 4. Key rotations KR

6 Keyed Permutation

The inspiration for the use of keyed permutation comes from the salt value in the Unix password encryption function `crypt(3)`, designed by Ken Thompson and Denis Ritchie at AT&T Bell laboratories. This function uses a fixed, publicly-known, 12-bit value to permute the DES E-box in the manner similar to that described in the ICE F function, except that it permutes only 24 of the 48 E-box output bits.

It was a small step to consider a modification where the salt was increased in size to permute all the bits from the E-box, and where the salt was not publicly known, but rather derived from the secret key in each round via some key schedule. It turns out that this has some useful cryptographic properties.

From an aesthetic point of view it has a certain consistency. The cornerstone of block cipher design has been the use of substitution and permutation networks to encrypt data [9]. Keyed substitution, usually in the form of XORing key bits with an S-box input, has been commonly used, and it is felt that keyed permutation should complement this nicely.

To begin with, keyed permutation gives ICE immunity to complementation weaknesses. These result from situations where key and plaintext bits are inverted, but cancel each other out at the inputs to the S-boxes, causing the S-boxes to produce identical outputs. However, if key bits are inverted in ICE, the S-boxes will receive their inputs from totally different bits, and thus not regularly produce identical outputs.

A great challenge to cryptosystem designers has been resistance to differential cryptanalysis. Although the keyed permutation used in ICE doesn't grant immunity to differential cryptanalysis, it does greatly reduce its effectiveness.

Differential cryptanalysis relies an attacker sending XOR differences to the inputs of S-boxes, where the S-boxes will then produces an XOR difference as output with some probability. But because these input bits are being permuted in ICE, the attacker does not know which S-box will receive the bit.

However, an attacker can exploit the symmetry of the keyed permutation. Think of the 32-bit input to the F function as having left and right 16-bit halves. The permutation in ICE simply swaps bits between the halves. By using input differences where both halves are the same (called *symmetric* inputs), an attacker can be certain that the bits will reach their target S-boxes.

However, since the attacker now has to target at least two S-boxes at a time, the probability of success is typically squared or worse. Detailed values are given in later sections.

Similarly, linear cryptanalysis is forced to use symmetric inputs, again with much lower probabilities.

7 The Design of the S-boxes

The S-boxes in ICE were primarily designed to be resistant to differential cryptanalysis, and in particular to symmetric attacks. In an ideal world, every possible Galois Field prime and XOR offset would be tried, and the resulting S-boxes evaluated. However, with 30 Galois primes to try in 16 positions, and 16 8-bit offsets to choose, this would require more than 10^{71} evaluations. As result, the selection process was done in incremental steps.

1. Of the ten bits input to each S-box, the middle six bits are unique to that S-box, while the remaining four are shared with adjacent S-boxes. Since each row only uses one prime, XOR profiles for all 30 different primes were generated, and groups of four primes were evaluated by adding their profiles together and checking the probabilities of characteristics whose inputs use the middle six bits. 509 groups of four were found whose peak probability was 18/1024.
2. Pairs of these four-prime sets were evaluated, with the proviso that no pair could share a prime in common. The aim was find the pair with the lowest product of probabilities given the same input probability. The pairs chosen appeared to have probability products of 256/1048576, but it later turned out that the analysis software was flawed, and the true probability was 324/1048576. However, by then the design was complete, and the problem was not deemed serious enough to justify a redesign.
3. The XOR offsets in each S-box were selected to minimise the probability of an input difference producing a zero output difference, and consequently to minimise the probability that a symmetric input difference would produce a zero output difference.

4. 4096 sets of values had the same probabilities ($4320/2^{40}$), so the set was chosen which had the lowest probability of a symmetric input difference producing itself as output ($8064/2^{40}$).

With respect to differential cryptanalysis, it makes no difference if all the XOR offsets in a row are XORed with a constant, or if the positions of the primes are XORed with a constant, so there were still 1024 choices available for each S-box. None of these choices would have any effect on differential (and, it turns out, linear) cryptanalysis, so some new, and fairly arbitrary, criteria were used to narrow down the choice of parameters.

1. There should be no x where $F(x) = 0$, assuming a zero subkey. If F in all rounds produces zero outputs, then a plaintext would encrypt to itself.
2. There should be no x where $F(x) = x$.
3. For a given input, no S-box shall produce the same output as another S-box.
4. The sum of the bit count of $F(x) \oplus x$ over all symmetric x values shall be perfectly balanced (i.e. equal to 1048576).
5. Finally there were 16 sets of values to choose from. They were evaluated for the bitcount of $F(x)$ for all 32 single-bit x values. None gave the balanced value of 512, but the closest was 506, which was then chosen as the set of primes and offsets for the S-boxes.

8 Cryptanalysis

During the design process, ICE was subjected to a number of attacks. In addition to the key schedule analysis described previously, it was also subjected to differential cryptanalysis, linear cryptanalysis, and some other specialised attacks.

8.1 Differential Cryptanalysis

There are 22 one-round characteristics with probability 18/1024. However, they require non-symmetric inputs, so are not effective attacks.

The two best symmetric single round characteristics are $F(008c008c) \rightarrow 4042a085$ and $F(00dc00dc) \rightarrow 5920a681$, both with probability 324/1048576. However, because they are not symmetric they cannot be turned into iterative characteristics.

For characteristics with symmetric inputs and outputs, the best examples are $F(b801b801) \rightarrow 02b702b7$ with probability $57600/2^{40}$ and $F(34eb34eb) \rightarrow d82fd82f$ with probability $50176/2^{40}$.

The best symmetric characteristics where the output difference equals the input are $F(80848084) \rightarrow 80848084$ and $F(98619861) \rightarrow 98619861$, both with probability $8064/2^{40}$. Although these can be turned into an iterative attack, the probabilities are too low to be useful.

The best symmetric characteristics that produce zero output differences are $F(b2d6b2d6) \rightarrow 0$ and $F(cad6cad6) \rightarrow 0$, both with probability $4320/2^{40}$

Both of these characteristics can be turned into a five-round characteristic with a probability of $2^{-55.85}$. It is possible that this attack could be used to break Thin-ICE in less time than exhaustive search, although this has not been investigated fully at present.

The full 16-round ICE and all its extended variants appear to be secure against diffferential cryptanalysis.

8.2 Linear Cryptanalysis

Linear cryptanalysis relies on correlations between the XOR sum of certain bits in the input and output of S-boxes. However, the use of keyed permutation means that, although the output of the S-boxes are usable in the same way as the DES attack, the source of the input bits cannot be known for certain unless symmetric input bits are used.

For ICE, the best approximation for a single S-box is $NS_2(457, 136) = 416 = 512 - 96$. However, since this is not symmetric it is not usable as an attack.

Linear approximations that only make use of the two bits shared by adjacent S-boxes do not need knowledge of input bits, and can be combined into iterative expressions. The best approximations of this form are ...

$$X[24,25] \oplus X[24,25] \oplus F(X,K)[11,12,18,20,25,31] \oplus F(X,K)[4,9,17,22,27] = K[28,29] \oplus K[30,31]$$
$$X[8,9] \oplus X[8,9] \oplus F(X,K)[3,8,16,26] \oplus F(X,K)[13,19,21,30] = K[8,9] \oplus K[10,11]$$
$$X[8,9] \oplus X[8,9] \oplus F(X,K)[3,14,23,26] \oplus F(X,K)[19,21,30] = K[8,9] \oplus K[10,11]$$

Each of these linear approximations has a probability of $2^{-7.83}$. They can be combined into a symmetric 6-round expression which has a probability of 2^{-42}. If used to attack Thin-ICE it would typically require 2^{82} ciphertexts to achieve a 75% success rate, so it appears that Thin-ICE cannot be broken using this attack.

8.3 Other Attacks

It is possible that the use of keyed permutation introduces weaknesses of its own. One possibility is to somehow trace the path of bits through the cipher, and thus deduce where they were permuted. This would immediately yield key bits. However, early analysis indicates that the avalanche effect of the S and P boxes masks any information of this sort after a few rounds.

For extended-round variants of ICE with keys 128 bits and longer, it must be remembered that the strength of the cipher under a chosen-plaintext attack is only 2^{64} time and memory, since an attacker can theoretically simply store all the plaintext/ciphertext pairs in one massive lookup table, and thus immediately find the plaintext corresponding to a ciphertext. Although this may not be a practical attack, it does represent the theoretical strength of the cipher under chosen-plaintext attacks, and, to a lesser extent, known-plaintext attacks.

9 Software Implementation

Keyed permutation is achieved by simple bitwise operations. This is best demonstrated in the ICE F function source code, written in ANSI C. Note that for speed the S-boxes have been pre-permuted to produce 32-bit outputs. Full source code implementations of ICE, written in ANSI C, C++, and Java, can be found at [6].

```
unsigned long
ice_f (
    unsigned long           p,
    const ICE_SUBKEY        sk
) {
    unsigned long   tl, tr;   /* Expanded 40-bit value */
    unsigned long   al, ar;   /* Salted expanded 40-bit value */

                              /* Left half expansion */
    tl = ((p>>16) & 0x3ff) | (((p>>14) | (p<<18)) & 0xffc00);

                              /* Right half expansion */
    tr = (p & 0x3ff) | ((p<<2) & 0xffc00);

                              /* Perform the keyed permutation */
    al = sk[2] & (tl ^ tr);
    ar = al ^ tr;
    al ^= tl;

    al ^= sk[0];              /* XOR with the subkey */
    ar ^= sk[1];

                              /* S-box lookup and permutation */
    return (ice_sbox[0][al>>10] | ice_sbox[1][al & 0x3ff]
            | ice_sbox[2][ar>>10] | ice_sbox[3][ar & 0x3ff]);
}
```

The C implementations were bechmarked against an optimised version of DES on a 100MHz 486 PC running Linux, as shown in table 5.

The slow speed of key changes was surprising, given the simplicity of the key scheduling algorithm. It turns out this is because the algorithm operates on only one key bit at a time, whereas this DES implementation operates on 28-bit blocks.

10 Summary

The design and analysis of ICE and its variants has been described here. It is hoped that the ciphers will prove secure in the long run, and provide possible

Operation (x 100000)	Time (seconds)
DES encryption	2.37
DES decryption	2.40
DES key change	4.98
ICE encryption	1.63
ICE decryption	1.59
ICE key change	44.79
Thin-ICE encryption	0.88
Thin-ICE decryption	0.87
Thin-ICE key change	22.45
ICE-2 encryption	3.12
ICE-2 decryption	3.04
ICE-2 key change	89.63

Table 5. Benchmark results

Variant	Key	Plaintext	Ciphertext
ICE	`deadbeef01234567`	`fedcba9876543210`	`7d6ef1ef30d47a96`
Thin-ICE	`deadbeef01234567`	`fedcba9876543210`	`de240d83a00a9cc0`
ICE-2	`00112233445566778899aabbccddeeff`	`fedcba9876543210`	`f94840d86972f21c`

Table 6. Certification triplets

alternatives for DES in the future. As a public-domain algorithm, with source code freely available (export restrictions permitting), it should be useful in any number of security and privacy applications.

References

1. E. Biham, New Types of Cryptanalytic Attacks Using Related Keys, *Advances in Cryptology - EUROCRYPT '93 Proceedings*, Springer-Verlag, pp. 386-397, 1994
2. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993
3. L. Brown, J. Pieprzyk and J. Seberry, LOKI: A Cryptographic Primitive for Authentication and Secrecy Applications, *Advances in Cryptology - AUSCRYPT '90 Proceedings*, Springer-Verlag, pp. 229-236, 1990
4. M.E. Hellman, R.C. Merkle, R. Schroeppel, L. Washington, W. Diffie, S. Pohlig and P. Schweitzer, Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard, Technical Report SEL 76-042, Stanford University, September 1976
5. M. Kwan and J. Pieprzyk, A General Purpose Technique for Locating Key Scheduling Weaknesses in DES-Like Cryptosystems, *Advances in Cryptology - ASIACRYPT '91 Proceedings*, Springer-Verlag, pp. 237-246, 1991

6. M. Kwan, The ICE Home Page, http://www.cs.mu.oz.au/~mkwan/ice
7. M. Matsui, Linear Cryptanalysis Method for DES Cipher, *Advances in Cryptology - EUROCRYPT '93 Proceedings*, Springer-Verlag, pp. 386-397, 1994
8. National Bureau of Standards, *Data Encryption Standard*, FIPS PUB 46, U.S. Department of Commerce, 1977
9. C.E. Shannon, Communications Theory of Secrecy Systems, *Bell System Technical Journal*, vol. 28, no. 10, pp. 656-715, October 1949
10. W. Tuchman, Hellman Presents No Shortcut Solutions to DES, *IEEE Spectrum*, v. 16, n. 7, pp. 40-41, July 1979
11. P.C. van Oorschot and M.J. Weiner, A Known-Plaintext Attack on Two-Key Triple Encryption, *Advances in Cryptology - EUROCRYPT '90 Proceedings*, Springer-Verlag, pp. 318-325, 1991