Evolución de los algoritmos de bloque luego de DES: (algunos highlits):

1) GOST 28147-89: Feistel, pero con 32 rondas y clave de 256 bits. La funci'on de ronda no tiene expansión, sino simplemente SUMA con la clave, particion en nibbles y pasado de cada nibble por un S-box 4 por 4. La permutación de bits es simplemente la rotación a izquierda por 11 bits de la palabra de 32 bits resultante. Historia: URSS, fines de los 70s, Top Secret. Bajo a solo Secret en 1990, y justo despues de la disolución de la URSS, se hizo publico. Los S-boxes no eran fijos pero no dependian de la clave: mas bien, cada implementación tenia sus S-boxes. Probabalemente no seguros. (el 1 al menos no es óptimo)

2) FEAL (Fast Encription Algorithm). Desarrollado en Japón a fines de los 90's. La idea era usar una función de ronda mas segura que la de DES, y poder reducir el numero de rondas, haciendolo mas rapido, pero fracasó. La función de ronda usaba dos "S-boxes" 16 por 8, pero en realidad no eran tales, simplemente se sumaban los dos bytes de entrada, y en uno de los S-boxes se le suma un 1 ademas, y luego al byte de salida se lo rotaba 1 bit. Por lo tanto, en realidad era un cifer sin S-boxes , solo con xors, sumas, y rotaciones, y fue facilmente atacado con DC y LC.

3) IDEA: (International Data Encription Algorithm) Es un cifer de bloque de 64 bits y calve de 128 bits. Aqui tampoco se usan S-boxes, pero si operaciones de grupos incompatibles entre si. Las operaciones son: XOR, suma modulo 2^{16}, y el producto en $\mathbb{Z}\!\!\mathbb{Z}^*_{2^{16}+1}$ (mirando al 0 como 2^{16}). Denotando este producto por $\odot$, IDEA es: dado un bloque de 4 palabras X_i de 16 bits c/u, 8 rondas todas como la siguiente:

.-Al principio de la ronda, mezclar el bloque con la clave, usando la operación del grupo $\mathbb{Z}\!\!\mathbb{Z}^*_{2^{16}+1} \times \mathbb{Z}\!\!\mathbb{Z}_{2^{16}} \times \mathbb{Z}\!\!\mathbb{Z}_{2^{16}} \times \mathbb{Z}\!\!\mathbb{Z}^*_{2^{16}+1}$, es decir,

$$(X_1, X_2, X_3, X_4) = (X_1, X_2, X_3, X_4)(\odot, +, +, \odot)(K_1^r, K_2^r, K_3^r, K_4^r)$$

Luego, hacer $(F_1, F_2) = (X_1, X_2) \oplus (X_3, X_4)$ y usar (F_1, F_2) como entradas al MA-box, que consiste en:

$$\begin{aligned} F_1 =& F_1 \odot K_5^r \\ F_2 =& (F_2 + F_1) \odot K_6^r \\ F_1 =& F_1 + F_2 \end{aligned}$$

Luego de la salida del MA-box, hacer $X_i = X_i \oplus F_1,\ i = 2, 4$ y $X_i = X_i \oplus F_2,\ i = 1, 3$.

y luego de esto, hacer un twist: $(X_2, X_3) = (X_3, X_2)$. En la ultima ronda no hacer ese twist, y hacer un ultimo mix con clave, usando otra vez la operacion $(\odot, +, +, \odot)$.

La expansión de clave es sencilla: los 128 bits se dividen en 8 bloques de 16 bits, y estan son las primeras 8 claves (por ronda se usan un total de 6 claves). Luego se rotan los 128 bits a la izquierda 25 bits, y se extraen otras 8 claves, se vuelven a rotar 25 bits, etc.

La parte principal usa lo que se llama una estructura Lai-Massey: dado un bloque (L, R), y una función de ronda F que trabaja sobre la mitad de los bits, en vez de hacer como en Feistel $(L, R) = (L \oplus F(R), R)$, se calcula $A = F(L \oplus R)$ y luego se hace $(L, R) = (L, R) \oplus (A, A)$. Observemos que esta operación es invertible, pues si denotamos por (L^*, R^*) la salida, entonces $L^* \oplus R^* = (L \oplus A) \oplus (R \oplus A) = L \oplus R$, con lo cual recuperamos la entrada original a la función F y podemos recalcular A y recuperar (L, R) a partir de (L^*, R^*).

Esta propiedad permite ver que desencriptar con IDEA es igual que encriptar, solo que hay que cambiar el orden de las claves de ronda, y en el caso de las aditivas, tomar la clave opuesta, y en el caso de las multiplicativas, tomar las inversas. (salvo para las claves que se usan en el MA-box, esas quedan como estan). En definitiva, si las claves son $K_1^1, K_2^1, ..., K_6^8, K_1^9, K_2^9, K_3^9, K_4^9$, entonces las claves para invertir son, en orden:

$(K_1^9)^{-1}, -K_2^9, -K_3^9, (K_4^9)^{-1}, K_5^8, K_6^8$ para la primera ronda, y $(K_1^8)^{-1}, -K_2^8, -K_3^8, (K_4^8)^{-1}, K_5^7, K_6^7$ para la segunda, etc.

El problema con respecto a Feistel es que para seguir en Feistel basta con hacer el twist $(L, R) = (R, L)$ para seguir, y Ruby y Lackoff probaron que 3 rondas de Feistel con función de ronda aleatoria producen una permutación indistinguible de una permutación aleatoria. En cambio, con la estructura Lai-Massey eso no pasa, porque de hecho aunque hagamos el twist el xor de las mitades queda igual. Asi pues, hay qye hacerle alguna otra transformación a todo el bloque (L, R). En el caso de Idea, esa transformación consiste en partir una vez mas los bloques L y R, e intercambiar los sub-bloques intermedios. Aun asi, esa función de ronda

no seria muy segura, PERO, tenemos ademas la mezcla con la clave de TODO el bloque al principio de las rondas, lo cual añade un factor extra que, ahora si, la vuelve segura. Pero si por ejemplo se reemplazan los $\odot$ por $+$, entonces se esta en problemas. Ademas, como los $\odot$ solo se usan para multiplicar claves con datos, y nunca datos entre si, lo que puede pasar es que para algunas claves, la multiplicación no haga una buena tarea. De hecho, eso es exactamente lo que pasa, y IDEA tiene un gran conjunto de claves debiles, pero no debiles en el sentido de DES, sino debiles en el sentido que esas claves vuelven la operación $\odot$ lineal o casi con respecto ya sea a la suma o al xor. El número de clases débiles que se encontraron originalmente (en 1993) era enorme (2^{51}) la probabilidad de usar una es 2^{-75}. Luego se encontraron mas claves débiles, pero siguen siendo una fracción del total. Una forma de evitar por lo menos las primeras es descartar cualquier clave que tenga 15 o mas ceros consecutivos. (parte del problema radica en la extremada sencillez de la expansión de clave de IDEA).

El mejor ataque conocido contra IDEA solo puede atacar 4 rondas mas la mezcla final (4.5 rondas) de las 8.5 que tiene IDEA, usando 2^{64} textos y 2^{112} encripciones. Algunas variantes de IDEa son muy inseguras, por ejemplo reemplazar la mezcla con clave usando $(\odot, \oplus, \oplus, \odot)$ en vez de $(\odot, +, +, \odot)$ puede quebrarse con solo 2^{30} (chosen) plaintexts.

4) Khuffu: aca se usaron por primera vez S-boxes grandes, 8 por 32. Los S-boxes eran secretos y dependian de la clave. Basicamente, era un Feistel en el cual se elejian 8 bits de una mitad, se pasaban por el S-box para obtener 32 bits, y se xoreaba el resultado a la otra mitad. Habai algunos shifts para asegurar que no se usaran siempre los mismos 8 bits. Khafre, que era basicamente Khufu con S-boxes no secretos, era facilmente quebrable. Tambien se introdujo por primera vez el whitening, auynque la palabra la inventó Rivest mas tarde para DES-X.

5) CAST: siguió con la idea de los S-boxes 8 por 32, pero mejor: ahora se usaban todos los bytes de la parte que se procesa,, no solo uno, y lo que se hacia era pasar cada byte por un S-box distinto, y las 4 palabras de 32 bits luego xorearlas entre si, antes de xorearlas a la otra mitad del cifer. Los S-boxes eran fijo, pero desarrollados para resistir DC y LC. Otras versiones posteriores usaron cosas mas complicadas con esas 4 mitades. (se inspiraron en Blowfish,ver mas abajo) Por ejemplo, CAST-128 (de 64 bits, con clave de 128), xoreaba las dos primeras palabras de 32 bits, al resultado le restaba la tercera, y luego sumaba la cuarta, antes de xorear con la otra mitad. Eso hacia en las rondas congruentes a 1 modulo 3. En las rondas congruentes a 2, primero restaba, luego sumaba, luego xoreaba, y en las congruentes a 0, sumaba, xoreaba, restaba.

6) Blowfish: Schenier diseño este algoritmo, probablemente uno de los mas seguros de 64 bits. Era un algoritmo parecido al CAST original, solo que en vez de xorear todo, sumaba las dos primeras palabra, al resultado lo xoreaba con la tercera, y luego sumaba la cuarta. Pero lo que hacia a Blowfish tan seguro es que los S-boxes no eran fijos sino dependientes de la clave. ¿Como se generaban? Las claves de ronda (que se xorean) y los S-boxes se llenaban primero con los digitos hexadecimales de π. Luego se tomaba la clave y se la xoreaba a todas las claves de ronda y de whitening final. Con estas claves, se encripta el bloque 0. El resultado final se usa para reemplazar las dos primeras claves de ronda, y con esta modificacion, ahora se encripta el resultado que se habia obtenido. El resultado se usa para reemplazar las dos siguientes claves de ronda, etc, hasta agotar todas las claves de ronda, y luego se sigue, reemplazando de esta manera las entradas de los S-boxes.

Blowfish es muy rapido en software, pero tiene obviamente un tiempo grande de expansion de la calve (un total de 521 iteraciones del algoritmo hasta que se pueda encriptar con el), por o que no es recomendable si se debe usar en una aplicación que deba cambiar claves frecuentemente.

7) DES-x: Este algoritmo fue propuesto por Rivest para superar el problema de las claves cortas de DES: consiste en usar una clave de 184 bits, simplemente xoreando 64 bits de clave al principio, usar DES con otros 56 bits, y luego xorear otros 64 bits de clave al final. Si bien esto no provee practicamente ninguna protección extra contra DC o LC, si lo hace contra busqueda de claves exhaustiva.

8) Para correjir el problema de la clave, lo mas usado es doble o triple encripción. Doble encripción resulta no ser tan mas segura, y se recomienda triple. Doble no es tan segura por un Meet in the Middle attack: uno, dado un para (P_1, C_1), encripta con todas las claves posibles P_1 y lo guarda en una tabla. Una vez terminado esto, para todas las claves posibles va desencriptando C_1 y chequeando el valor obtenido contra la tabla. Si uno obtiene un valor, la clave que indexa esa entrada en la tabla es probablemente la correcta. Para verificarla, uno la chequea en otro par (P_2, C_2) (para una probabilidad de un falso positivo

de solo 2^{-16}, o bien contra otros dos pares de textos (aca la probabilidad de un falso positivo es 2^{-80}).

Con triple encripción se sugirió usar la modalidad de usar solo dos claves: es decir, que la primera y tercera clave sean iguales. En este caso tambien hay un ataque, pero ya no es de known pltxt sino de chosen:

Para todas las claves i, encriptar 0 con esa clave y guardar el resultado en una tabla T. Al mismo tiempo, desencriptar 0, obteniendo un pltxt P_i, pedir que lo encripten con las claves que se estan atacando, obtener C_i, desencriptar C_i con i, y obtener B_i y guardarlo en una tabla B. Luego chequear las tablas B y T hasta encontrar (i,j) con $T_j = B_i$, y chequear que esta sea la clave con algun otro par pltxt,cphtxt.

Por eso, se recomienda triple encription con clave triple, y se usa generalmente el modo EDE, es decir, la encripcion del medio es una desencriptacion, para que al hacer las tres claves iguales quede backward compatible con single DES.

AES: Fialmente, en 1997-8, se llamó a un concurso público para encotnrar al sucesor del DES. PERO, entre las especificaciones se pedia un algoritmo de bloque de 128 bits, con lo cual todos los anteriores (y otros que no nombre) quedaron subitamente viejos. La clave debia ser de 128, 192 o 256 bits. Se presentaron 15 candidatos, algunos muy malos (el presentado por Alemania fue quebrado en dias cuando se juntaron varios de los otros teams para analizarlo). Luego de algun tiempo, se eligieron 5 finalistas, y finalmente un ganador. Esos son los que estudiaremos luego.