Il Blowfish per la crittografia dei dati

In questo articolo si esamineranno le modalità di utilizzo dell'algoritmo Blowfish per la protezione delle informazioni nei sistemi embedded

Francesco Pentella

Oggi come non mai, la sicurezza dei dati è un argomento di sicuro interesse. In particolare per i dispositivi collegati in rete o, comunque, quando le informazioni sono condivise in una varietà di applicazioni.
L'esigenza di avere sistemi protetti nasce quando si vuole rendere riservato il contenuto di informazioni che, oggettivamente, si considerano sensibili.
Sistemi di questo genere certamente implicano una codifica dei dati tale per cui vengono resi incomprensibili a terzi. Inoltre, occorre prevedere anche un meccanismo che garantisca l'integrità dei dati e che consenta di autenticare la sorgente. Per questa ragione si introducono delle funzioni crittografiche. Un algoritmo di crittografia è un insieme di regole logiche e matematiche utilizzate per convertire informazioni (testo in chiaro) da una forma leggibile in una forma cifrata (testo cifrato o ciphertext).
L'algoritmo utilizzato per la codifica delle informazioni è una funzione matematica applicata al messaggio originale e che produce un testo cifrato; l'operazione inversa è l'operazione di decodifica, cioè una funzione matematica applicata al testo cifrato per ottenerne uno in chiaro. La figura 1 mostra queste operazioni.
Il processo di codifica e decodifica può essere effettuato solo se si conoscono:
- il principio di funzionamento dell'algoritmo;
- la chiave utilizzata associata al processo di codifica e decodifica.
Sicurezza e crittografia sono due concetti diversi. La crittografia tratta il problema della segretezza delle informazioni, mentre la sicurezza utilizza strumenti crittografici per realizzare applicazioni robuste alla presenza di attacchi condotti da avversari.
Un sistema del genere deve soddisfare i seguenti requisiti di sicurezza:
- Disponibilità. Questo implica rendere disponibili a ciascun utente abilitato le informazioni a cui ha diritto di accedere, nei tempi e nei modi previsti.
- Riservatezza. Nessun utente deve poter ottenere o dedurre dal sistema informazioni che non è autorizzato a conoscere.
- Integrità. Impedire l'alterazione diretta o indiretta delle informazioni, sia da parte di utenti e processi non autorizzati, sia a seguito di eventi accidentali.
- Autenticazione. Ciascun utente deve poter verificare l'autenticità delle informazioni.
- Non ripudiazione. Nessun utente deve poter ripudiare o negare messaggi da lui spediti o firmati. Evitare che le informazioni e i messaggi siano negati dal firmatario in tempi successivi (per esempio la firma di un contratto).

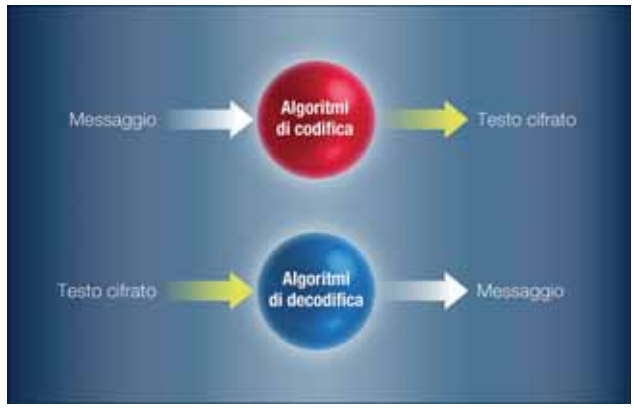

Fig. 1 - Processo di decodifica e codifica

Gli algoritmi di crittografia sono classificati in due grandi famiglie. Quando la chiave utilizzata per la codifica coincide con quella di decodifica lo schema crittografico si definisce di tipo "simmetrico" (Fig. 2). La chiave condivisa è indicata come chiave comune. Viceversa, se la chiave usata nel processo di decodifica è diversa l'algoritmo si definisce "asimmetrico" o "a chiave pubblica e privata" (Fig. 3).
Il Blowfish utilizza un'unica chiave segreta per la fase di codifica e per quella di decodifica; per questa ragione è definito come algoritmo "simmetrico" o "a chiave comune". In questo

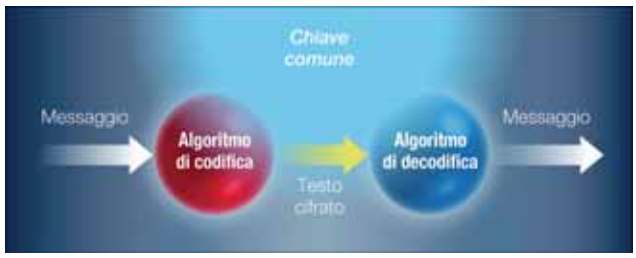

Fig. 2 - Chiave comune

modo i due personaggi di un mondo fittizio, Alice e Bob, comunicano in maniera riservata attraverso la procedura come riportata in figura 4. Dalla figura si evince che:
- Alice e Bob selezionano un algoritmo comune;
- Alice e Bob stabiliscono una chiave segreta;
- Alice, dopo aver generato il messaggio in chiaro, lo codifica utilizzando la chiave segreta;
- Alice, a questo punto, invia il messaggio cifrato a Bob;
- Bob decodifica il messaggio utilizzando la chiave comune.

Algoritmo Blowfish

Il Blowfish è un cifrario a blocchi a chiave simmetrica, sviluppato da Schneier e pubblicato nel 1994. L'idea di Schneier era proporre una valida alternativa a DES. Oggi questo algoritmo vanta molte realizzazioni ed è utilizzato in un'infinità di prodotti, probabilmente grazie anche alla sua caratteristica di non essere protetto da brevetti. A tale proposito Schneier dichiarò: "Blowfish è libero da brevetti, e rimarrà tale in tutte le nazioni. L'algoritmo è di pubblico dominio e può essere usato liberamente da chiunque". Blowfish risulta essere più veloce del DES quando viene implementato su microprocessori a 32 bit con una grossa cache dati, come il Pentium e il PowerPC.
Questo algoritmo utilizza varie tecniche tra cui la rete Feistel, le S-box dipendenti da chiavi e funzioni F non invertibili. Le chiavi utilizzate sono di dimensioni variabili, fino a un massimo di 448 bit, e i blocchi utilizzati per la cifratura sono di 64 bit.
Questo tipo di algoritmo è anche chiamato "a chiave segreta", poiché richiede che i due protagonisti in gioco, Alice e Bob,

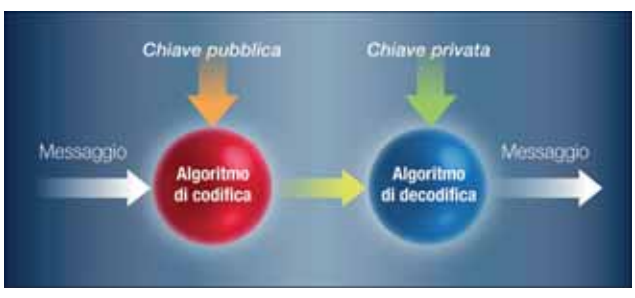

Fig. 3 - Crittografia a chiave pubblica e privata

utilizzino lo stesso algoritmo e la stessa chiave. Per questa ragione gli algoritmi di questo tipo richiedono mezzi trasmessivi sicuri per consegnare a ciascun attore la chiave. Esiste un limite importante: se la chiave è un elemento statico, cioè un numero concordato tra due attori, allora il numero delle chiavi cresce in maniera esponenziale con il numero degli attori coinvolti. Infatti, la relazione che lega il numero delle chiavi con gli attori è:

$N(N-1)/2$

Dove N identifica il numero degli attori coinvolti. Dalla relazione sopra mostrata si pone in evidenza che per 5 attori sono necessari 10 chiavi, mentre per 12 attori le chiavi ammontano a 66. Un sistema del genere implica che ogni attore debba con-

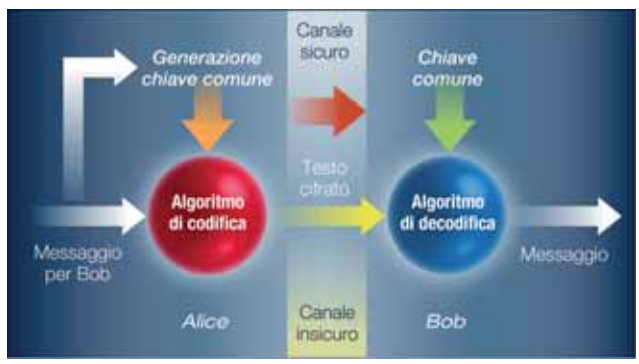

Fig. 4 - Schema per la crittografia simmetrica

servare N-1 chiavi. Nelle realizzazioni pratiche si tendono a utilizzare chiavi non statiche, ma si utilizzano sistemi automatici per generare e scambiare le chiavi.
Un algoritmo a chiave simmetrica non risolve il problema dell'autenticazione del messaggio o del suo rigetto qualora non risulti consono, ma conferisce solo riservatezza al flusso dei dati che saranno trasferiti. In questo senso non esiste una relazione univoca tra una chiave e un attore coinvolto; per questa ragione non è possibile attribuire in maniera precisa il messaggio spedito da un utente. Un algoritmo di questo tipo deve soddisfare alcuni requisiti, riassunti di seguito:
- il testo cifrato deve essere funzione di tutti i bit della chiave e del testo cifrato;
- non deve esserci nessuna relazione statistica tra testo in chiaro e testo cifrato;
- la modifica, anche di un singolo bit, nel testo o nella chiave presuppone la stessa probabilità di modifica del testo cifrato;
- di conseguenza la modifica di un bit nel testo cifrato comporta la stessa modifica nel testo decifrato.

Blowfish opera su di un blocco alla volta e le operazioni sono definite operazioni elementari. Il Blowfish ha la particolare caratteristica di essere abbastanza efficiente ed è un ottimo candidato per le soluzioni embedded; utilizza operazioni sem-

plici quali OR-esclusivo, addizioni, tabelle precalcolate e moltiplicazioni basate su modulo. Inoltre, le risorse fisiche richieste da Blowfish sono abbastanza contenute; si calcola che può richiedere alcuni Kbyte di ROM e poche decine di byte di RAM.
Blowfish è un cifrario a blocchi con chiave segreta a lunghezza variabile. La taglia del blocco è di 64 bit e la chiave può avere una lunghezza da 32 fino 448 bit. La fase di inizializzazione è abbastanza complessa, ma è eseguita prima della codifica. Lo schema dell'algoritmo Blowfish è mostrato in figura 5.
L'algoritmo Blowfish è diviso in due fasi:
A. Espansione della chiave. In questa fase si converte una chiave, che può arrivare fino a 448 bit, in vettore di sottochiavi per un totale di 4168 byte. Una particolarità di questo algoritmo è che l'espansione è effettuata usando Blowfish stesso.
B. Cifratura dei dati. Questo procedimento di ottiene attraverso una Feistel network di 16 round. Ciascun round consiste in una permutazione e in una sostituzione dipendente dalla chiave e dai dati utilizzando Blowfish stesso. Le sole operazioni usate sono le operazioni di XOR e addizioni di parole di 32 bit e quattro letture per round in array di dati indicizzati.
L'algoritmo Blowfish utilizza una

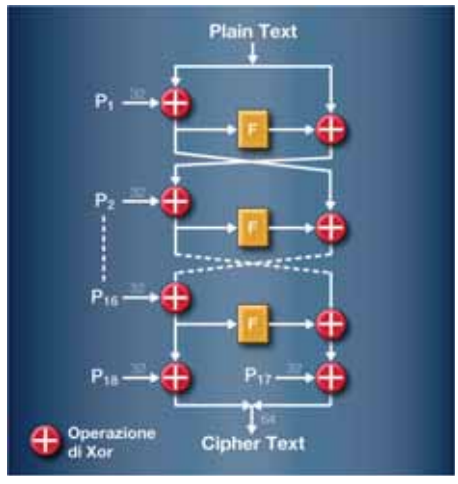

Fig. 5 - Schema algoritmo Blowfish

Listato 1
Algoritmo Blowfish in linguaggio C

```
#define N 16

typedef struct {
       uint32_t P[16 + 2];
       uint32_t S[4][256];
} BLOWFISH_CTX;

unsigned long F(BLOWFISH_CTX *ctx, uint32_t x)
{
uint16_t a, b, c, d;
uint32_t y;

       d = x & 0x00FF;
       x >>= 8;
       c = x & 0x00FF;
       x >>= 8;
       b = x & 0x00FF;
       x >>= 8;
       a = x & 0x00FF;
       y = ctx->S[0][a] + ctx->S[1][b];
       y = y ^ ctx->S[2][c];
       y = y + ctx->S[3][d];
       return y;
}

Void Blowfish_Encrypt(BLOWFISH_CTX *ctx, uint32_t *xl, uint32_t *xr)
{
uint32_t Xl;
uint32_t Xr;
uint32_t temp;
int i;

       Xl = *xl;
       Xr = *xr;
```

serie di sottochiavi che devono essere pre-calcolate prima di ogni cifratura e decifratura.
A. Il P-array consiste di 18 sottochiavi di 32 bit.
B. Esistono quattro S-box di 32 bit di 256 ingessi ognuno.
In questo modo il P-array ha l'esigenza di disporre di 4168 byte (18*4 + (256*4)*4).

Generazione di sottochiavi e S-Box

Le sottochiavi sono calcolate utilizzando Blowfish in base al seguente metodo:
1. Inizializzare anzitutto nell'ordine il P-array ed eseguire le quattro S-Box con una stringa definita. Questa stringa consta dei primi digit esadecimali di π, meno l'iniziale 3 (Tab.1). In ogni caso, Schneier non vincola a usare le cifre di π (come in questo caso) ma consiglia di usare una stringa costante indipendente da Blowfish e possibilmente calcolabile "in loco".

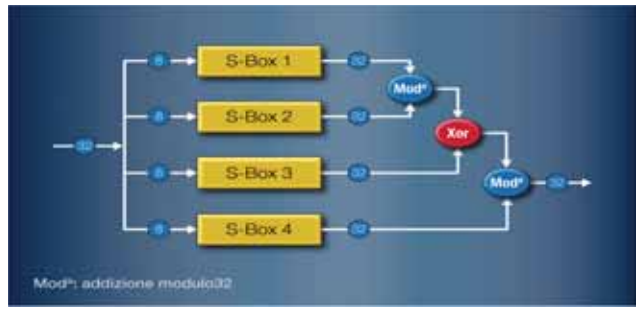

Fig. 6 - Funzione F

2. Fare l'operazione di xor fra i primi 32 bit della chiave k e P1, i secondi 32 bit con P2, … continuare, eventualmente fino a P14. Ripetere il passo 2 con i restanti Pi. Ripetere il ciclo attraverso i bit della chiave sino a che non sia stato eseguito uno XOR dell'intero P-array con i bit della chiave.

```
        for (i = 0; i < N; ++i)
                {
                        Xl = Xl ^ ctx->P[i];
                        Xr = F(ctx, Xl) ^ Xr;
                        temp = Xl;
                        Xl = Xr;
                        Xr = temp;
                }

        temp = Xl;
        Xl = Xr;
        Xr = temp;
        Xr = Xr ^ ctx->P[N];
        Xl = Xl ^ ctx->P[N + 1];
        *xl = Xl;
        *xr = Xr;
}

Void Blowfish_Decrypt(BLOWFISH_CTX *ctx, uint32_t *xl, uint32_t *xr)
{
uint32_t Xl;
uint32_t Xr;
uint32_t temp;
int i;

        Xl = *xl;
        Xr = *xr;
        for (i = N + 1; i > 1; --i)
                {
                        Xl = Xl ^ ctx->P[i];
                        Xr = F(ctx, Xl) ^ Xr;
                        temp = Xl;
                        Xl = Xr;
                        Xr = temp;
```

segue a pagina 72

3. Codificare 64 “0” con Blowfish e gli attuali Pi e Sj.
4. Sostituire P1 e P2 con l’output di Blowfish del passo 3.
5. Codificare l’output del passo 3 con Blowfish avente le prime due sottochiavi modificate.
6. Sostituire P3 e P4 con l’output di Blowfish del passo 5.
7. Continuare fino alla modifica di tutti i Pi e di tutte le S-Box.
8. In seguito, vengono sostituite ognuna delle quattro S-Box, ripentendo il procedimento fatto per le chiavi del P-array.

Poiché ogni iterazione genera due sottochiavi, allora sono necessarie 521 iterazioni per generare tutte le sottochiavi (18/2 iterazioni per generare il P-array e 256/2 iterazioni per ognuna delle quattro S-box).

Tabella 1 – P-array e S-Box

P-array	S-Box
P1 = 0x243f6a88	S1,0= 0xd1310ba6
P2=0x85a308d3	S1,1= 0xb8e1afed
P3=0x13198a2e	S1,2 = 0x24a19947
P4=0x03707344	—
P18 = 0x8979fb1b	S4,255 = 0x3ac372e6

Codifica

Questa parte consiste di 16 round. In ogni round viene attivata una permutazione dipendente dalla chiave e da una sostituzione dipendente dalla chiave e dai dati. Le operazioni utilizzate sono abbastanza semplici da essere abbastanza efficienti su microprocessori, quali XOR e addizioni di parole di 32 bit. La rete di Feistel è mostrata in figura 6 dove l’operazione di XOR denota un Xor logico bit a bit.

L’algoritmo di codifica è schematizzato nel modo che segue. Premessa: l’input è un blocco di 64-bit denotato con x. L’input x è diviso in due blocchi da 32-bit ciascuno (XL e XR):

```
                }

        temp = Xl;
        Xl = Xr;
        Xr = temp;
        Xr = Xr ^ ctx->P[1];
        Xl = Xl ^ ctx->P[0];
        *xl = Xl;
        *xr = Xr;
}

Void Blowfish_Init(BLOWFISH_CTX *ctx, uint16_t *key, int KeyLen)
{
uint32_t Xl;
{
int i, j, k;
uint32_t data, datal, datar;

        for (i = 0; i < 4; i++)
        {
                for (j = 0; j < 256; j++) ctx->S[i][j] = ORIG_S[i][j];
        }

        j = 0;
        for (i = 0; i < N + 2; ++i)
                {
                        data = 0x00000000;
                        for (k = 0; k < 4; ++k)
                        {
                                data = (data << 8) | key[j];
                                j = j + 1;
                                if (j >= keyLen) j = 0;
                        }
```

1. for i = 1 to 15
2. $x_L = x_L$ xor Pi
3. $x_R = F(x_L)$ xor x_R
4. Swap x_L and x_R
5. $x_L = x_L$ xor P16
6. $x_R = F(x_L)$ xor x_R
7. $x_R = x_R$ xor P17
8. $x_L = x_L$ xor P18
9. Ciphertext = x_L x_R

La funzione F esegue le seguenti operazioni, dalla figura 6:

- Dividere xL in quattro blocchi di 8 bit: a, b, c e d

- $F(x_L)=((S_{1,a}+ S_{2,b}$ modulo $2^{32})$ xor $S_{3,c})+S_{4,d}$ modulo $2^{32})$

Il listato 1 mostra una implementazione di Blowfish e risulta abbastanza contenuto da poter essere utilizzato per applicazioni di tipo embedded. Una realizzazione di questo tipo non richiede più di 6 Kbyte di memoria.
L'operazione di decodifica è identica a quella di codifica, eccetto che per l'uso inverso delle sottochiavi.

Prestazioni

L'algoritmo Blowfish consta di tre fasi: inizializzazione, codifica e decodifica. La fase di inizializzazione è la più dispendiosa in termini di risorse, infatti in questa parte occorre costruire il cosiddetto P-array e le S-Box.
Questo aspetto, per così dire negativo, viene comunque compensato dalle altre due fasi, che possono essere realizzate massimizzando tempo e risorse.

```
                        ctx->P[i] = ORIG_P[i] ^ data;
                }

        datal = 0x00000000;
        datar = 0x00000000;
        for (i = 0; i < N + 2; i += 2)
                {
                        Blowfish_Encrypt(ctx, &datal, &datar);
                        ctx->P[i] = datal;
                        ctx->P[i + 1] = datar;
                }

        for (i = 0; i < 4; ++i)
                {
                for (j = 0; j < 256; j += 2)
        {
                Blowfish_Encrypt(ctx, &datal, &datar);
                ctx->S[i][j] = datal;
                ctx->S[i][j + 1] = datar;
                }
        }
}

Int Blowfish_Test(BLOWFISH_CTX *ctx)
{
uint32_t L = 1, R = 2;

Blowfish_Init(ctx, (unsigned char*)"TESTKEY", 7);
Blowfish_Encrypt(ctx, &L, &R);
if (L != 0xDF333FD2L || R != 0x30A71BB4L) return (-1);
Blowfish_Decrypt(ctx, &L, &R);
if (L != 1 || R != 2) return (-1); return (0);
}
```