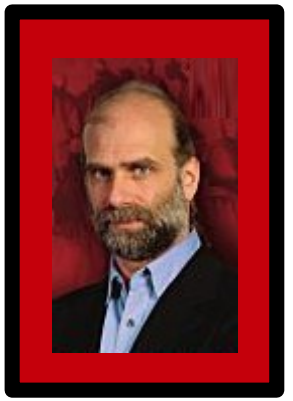
BLOWFISH

Un cifrario a blocchi di Bruce Schneier

Approfondimento del corso di Sicurezza
Anno Accademico: 2008/2009
A cura di: Andrea Marolla

Blowfish



Cifrario a blocchi (a chiave simmetrica).

Nasce nel 1993 da Bruce Schneier.

Motivo: valido sostituto a DES

Caratteristiche: libero da qualsiasi brevetto, di pubblico dominio e completamente disponibile → libertà di studiarne il codice.

Fonti: [1], [2]

Principali caratteristiche

- Basa il funzionamento su rete di Feistel a 16 rounds;
- Blocchi di 64 bits;
- Dimensione della chiave è variabile (fino a 448 bits);
- Due fasi: prima espansione della chiave, poi cifratura dei dati;
- Fa uso di sottochiavi ed SBOXes dipendenti dalla chiave primaria;
- E' molto veloce a patto di usare sempre la stessa chiave per cifrare.

Algoritmo	Clock cycles per round	# of round	# of clock cycles per byte encrypted
Blowfish	9	16	18
RC5	12	16	23
DES	18	16	45
IDEA	50	8	50
Triple-DES	18	48	108

Fonti: [1], [3]

Aspetti di design

- Manipolare dati in blocchi più grossi di singoli bit (32 bits);
- Usare blocchi di 64 bits (ma essere scalabile);
- Usare operazioni semplici efficienti su microprocessori (XOR, addizioni, tabelle, moltiplicazioni modulo);
- Poter essere implementato su processori con scarsa memoria;
- Permettere chiavi precomputabili (per eseguire le op. + velocemente);
- Permettere numero variabile di iterazioni;
- Non avere chiavi deboli;
- Usare sottochiavi che sono un hash “one-way” della chiave di partenza;
- Usare un design semplice da capire (uso di reti di Feistel);

Fonte: [1]

Cifrari a blocchi e reti di Feistel

Cifrari a blocchi moderni → “Iterated Ciphers”

Usano una “round function” ed un “key schedule”.

Lo scopo è quello di introdurre:

- Confusione (rendere complessa la relazione tra testo in chiaro e testo cifrato);
- Diffusione (distribuire la struttura statistica del testo per evitare attacchi statistici);

...attraverso permutazioni e sostituzioni...

Un cifrario di Feistel (iterated cipher) mappa un testo di 2t bits...

P = (L0, R0) → C = (Lr, Rr) con r-rounds.

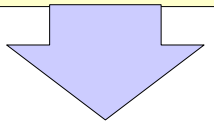
Round i-esimo: $(L_{i-1}, R_{i-1}) \xrightarrow{K_i} (L_i, R_i)$ $\quad L_i = R_{i-1} \quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

Vantaggi: Operazioni di cifratura/decifratura molto simili e cifrari facilmente comprensibili

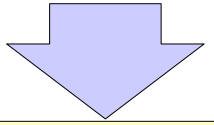
Fonti: [4], [5], [6], [7]

Fasi di Blowfish

Fase di generazione delle sottochiavi



...genera le sottochiavi per...



Fase di cifratura dei dati

Basata su rete di Feistel (16 round):
- operazione di permutazione dipendente da chiave;
- operazione di sostituzione dipendente da chiave e dati;

XOR e addizioni a 32bits

Fonte: [1]

Generazione delle sottochiavi

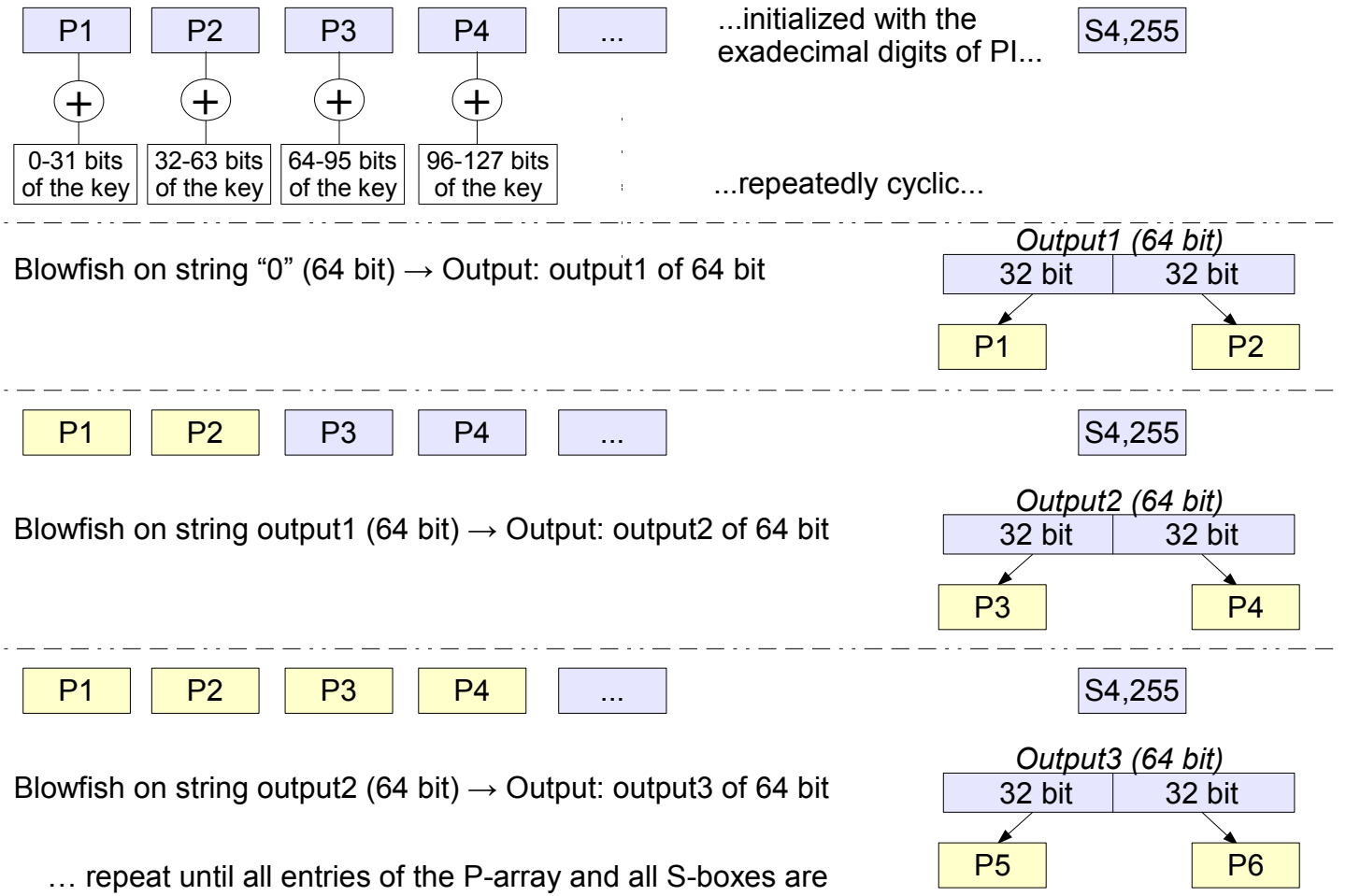
Si ricava:

- <u>P-Array</u>:
  `P1,..., P18`
  (32 bit ciascuno);

- <u>4 S-BOXes</u> ognuna costituita da 256 sottochiavi di 32 bit:

```
S1,0; ...; S1,255;
S2,0; ...; S2,255;
S3,0; ...; S3,255;
S4,0; ...; S4,255.
```

521 iterazioni per generare tutte le sottochiavi (9 per il P-array e 128*4 per le S-BOXes)



Fonte: [1]

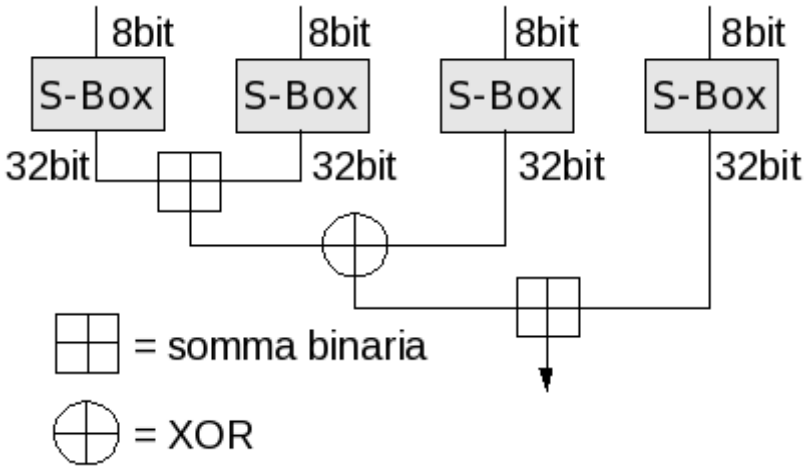
Fase di cifratura dei dati

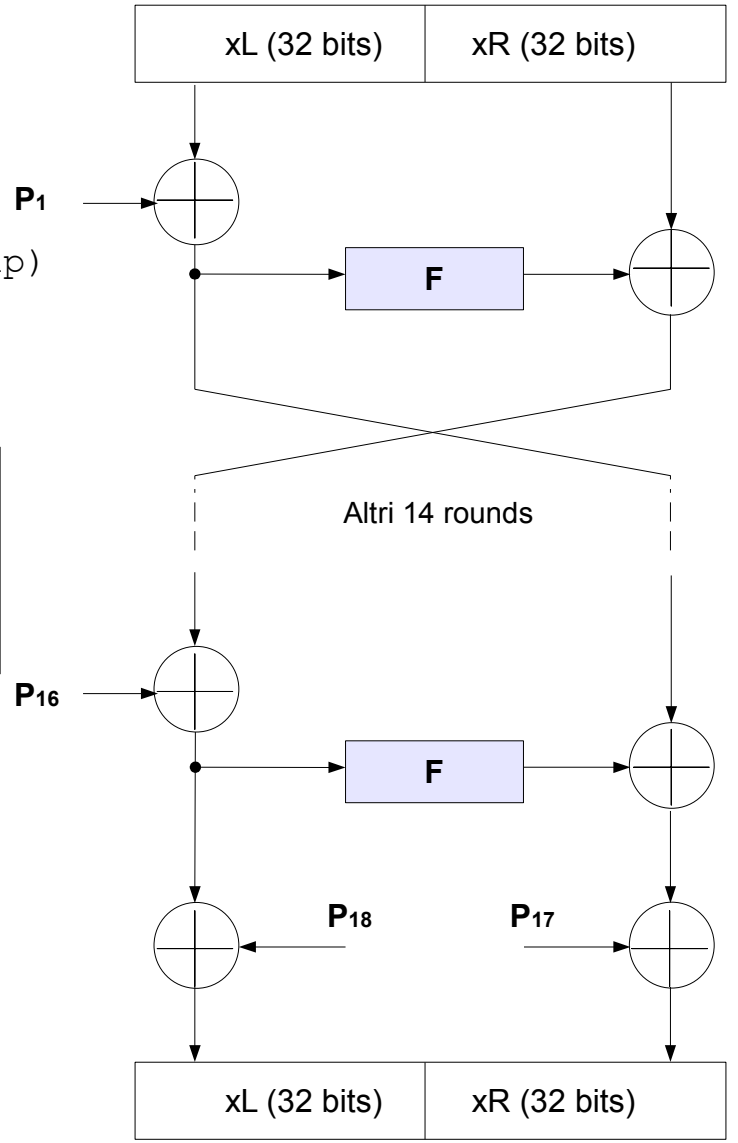
```
Dividi x in due parti di 32 bits xL, XR
FOR i=1 to 16:
     xL = xL ⊕Pi   (Pi = i-esima chiave)
     xR = f(xL) ⊕xR
     Swap xL and xR
     next i
Swap xL and xR (utile per annullare l'ultimo Swap)
xR = xR ⊕
xL = xL ⊕
Ricombina xL e xR
```

Funzione f

```
Si suddivide xL in
4 blocchi di 8 bits: a, b, c, d

F(xL) = ((S1,a + S2,b mod 2^32) ⊕ ,c)
                                + S4,d mod 2^32
```





Fonte: [1]

Sicurezza di Blowfish (1)

Blowfish fù pubblicato nel 1994. Il Dr. Dobb's Journal lancia un “contest” con premio $1000 per la migliore crittoanalisi di Blowfish ricevuta entro Aprile 1995.

I migliori risultati furono quelli di Serge Vaundenay che ha studiato degli attacchi “chosen plaintext” su una versione di Blowfish a 8 rounds.

Identifica la presenza di “weak keys” → K è debole se almeno una delle 4 S-BOXes ha una collisione.

Ad esempio se su S1 vi è una collisione significa che: $\exists a, a' (a \neq a') \mid S1, a == S1, a'$

Con “weak key” (conoscendo dunque le entry delle S-BOXes) è possibile conoscere P1, ..., P10 con 2^23 chosen plaintext (versione 16 rounds: 3 X 2^51).
Una chiave debole si ha mediamente ogni 2^15 chiavi.

Senza “weak key” sono necessari 2^48 chosen plaintext.
E' inoltre possibile scoprire se la chiave è debole (senza conoscere S-BOXes) con un attacco con 2^22 chosen plaintext.

Fonti: [1], [8]

Sicurezza di Blowfish (2)

Le sottochiavi del terzo e quarto round (P3 e P4) non dipendono dalla chiave principale.

1. P1, ...P18 inizializzati con i valori di π;
2. XOR per ogni Pi con i corrispondenti 32 bit della chiave;
3. Si cifra con Blowfish la stringa “0” ottenendo “x” di 64 bits;
4. **xL → P1; xR → P2;**
5. Si cifra l'output della fase 3 e si continua assegnando il risultato a P3 e P4...

PROBLEMA (a causa del punto 4)...

Prima iterazione di Blowfish:

```
XL = XL ⊕ P1 = XL ⊕ XL = 0;
```
(dunque XL diventa = 0!)

```
XR = XR ⊕ F(XL) = XR ⊕ F(0);
```
poi XOR con P2...

```
XR = XR ⊕ F(0) ⊕ P2 = XR ⊕ F(0) ⊕ XR = F(0);
```

Seconda iterazione:

```
XL = 0 ⊕ F(F(0)) = F(F(0));
```

Al termine XL → P3 e XR → P4, dunque P3 e P4 non dipendono più dalla chiave.

Fonte: [10]

Ricapitolando...

- ✔ Manipolare dati in blocchi più grossi di singoli bit (32 bits);
- ✔ Usare blocchi di 64 bits;
- ✔ Usare operazioni semplici efficienti su microprocessori (XOR...);
- ✔ Poter essere implementato su processori con scarsa memoria;
- ✔ Permettere chiavi precomputabili;
- ✔ Permettere un numero variabile di iterazioni;
- ✔ Usare un design semplice da capire (uso di reti di Feistel);

- ✔✗ Non avere chiavi deboli;
- ✔✗ Usare sottochiavi che sono un hash “one-way” della chiave di partenza;

Applicazioni di Blowfish

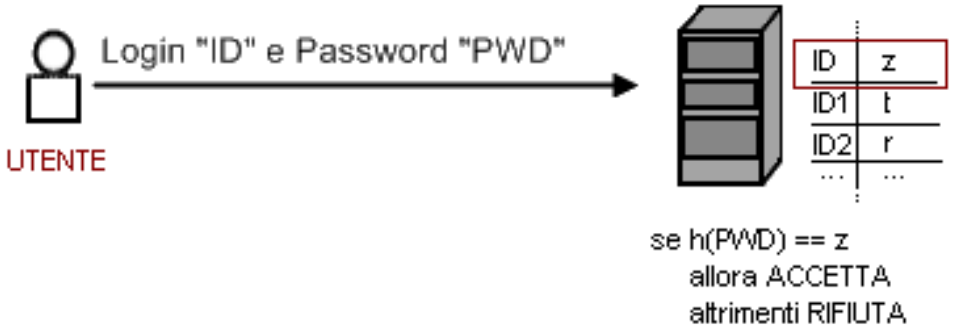
Blowfish è applicato in moltissimi casi reali. Nel sito di Schneier è presente una lista di tutte le applicazioni che lo utilizzano. Si possono citare:
- OpenBSD;
- Linux;
- PasswordSafe;
- Truecrypt;
- … oltre 150 applicazioni ...

Motivi del suo utilizzo sono:
- sicurezza della cifratura;
- velocità di cifratura (usando sempre la stessa chiave);
- ma anche “lentezza” di cifratura in caso di utilizzo di chiavi sempre diverse!

Fonte: [2], [11]

bcrypt: Blowfish in OpenBSD

Bcrypt: funzione “hash” di cifratura delle passwords.
Il sistema mantiene un file delle password, che contiene per ogni utente la sua password, cifrata opportunamente.



One-way: da y∈Y deve essere difficile computazionalmente trovare x !!!
Second preimage resistant: dato x, deve essere difficile trovare x'≠x tale che h(x)==h(x') !!!

In OpenBSD l'amministratore può scegliere lo schema di hashing delle password:
→ agendo sul file “passwd.conf” (prefissi usati: 1 → MD5 crypt; $2a$ → bcrypt).

Fonte: [12]

Attacchi al file delle Passwords

Attacco on line (S invia multiple richieste di login, provando tutte le possibili passwords...).

Difesa: rallentare le prove successive o limitarle;

Attacchi off-line (S attacca direttamente il file delle password, tentandole tutte o attraverso un “dictionary attack”.

Difesa: + complessa... uso di password rules, uso di salt...

Uso del salt... si memorizza nel file delle passwords... `UserID, salt, h(pwd, salt)`
In modo da proteggersi dagli attacchi basati su dizionario si genera random per ogni utente. Il salt deve avere una dimensione considerevole!!

- Nel 1976, con crypt (UNIX), si riusciva ad effettuare l'hash di al max. 4 passwords al sec;
- Dopo 20 anni → 200000 operazioni di crypt al sec. (gli attacchi si semplificano!)

- Oggi esistono sistemi più sofisticati del file delle password, ma questo continua ad avere grande utilizzo grazie anche alla sua semplicità.

Fonti: [7], [12]

crypt, MD5 crypt...

crypt nasce nel 1976, usa DES (passwords max. 8 cararatteri).

$$h(password) = \left(DES^{salt}_{56\text{bit}\, password}(0) \right) (25\, iterazioni)$$

Chiave del DES partendo dalla password di 8 caratteri:
Si prendono i 7 bits “low order” di ogni carattere della password.

Salt di 12 bits (stessa password per 4096 cifrature).
Salt modifica E di DES: scambio dei bits i e i+24 quando i è settato nel salt;

MD5 crypt evoluzione di crypt

La dimensione della password illimitata.
Salt di dimensione variabile (12 – 48 bits).

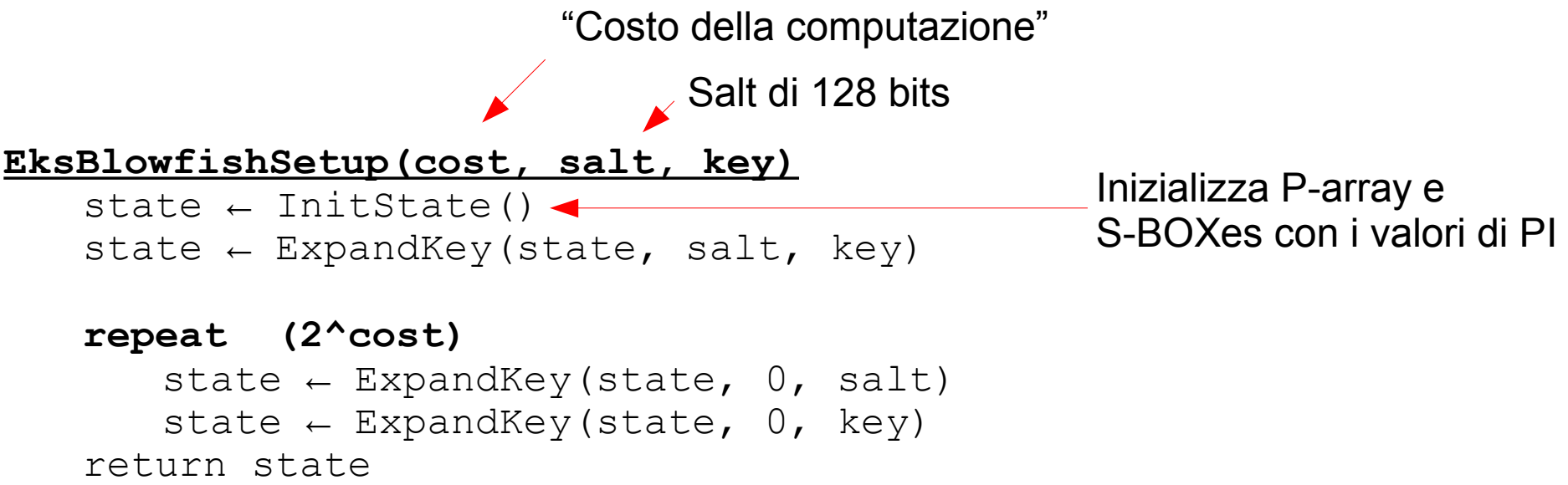
Non tengono parametrizzano però il costo della computazione!

Fonte: [12]

bcrypt

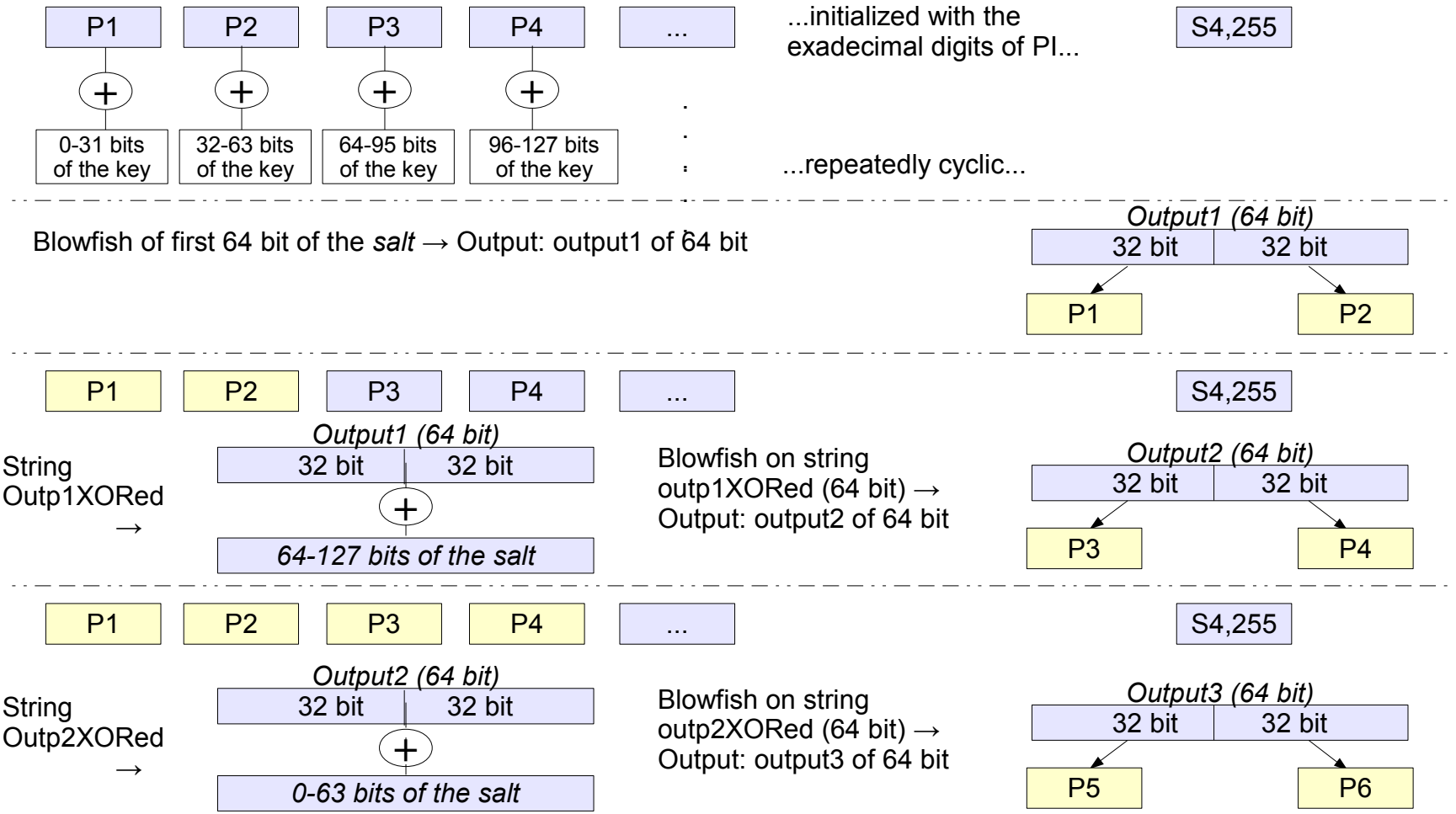
bcrypt parametrizza anche il costo della computazione...

`eksBlowfish` (esattamente come Blowfish), varia solo la generazione del P-array e delle S-BOXes, che vengono generate da `EksBlowfishSetup`.



Fonte: [12]

bcrypt: ExpandKey(state, salt, key)



Fonte: [12]

bcrypt: iterazioni di ExpandKey

```
EksBlowfishSetup(cost, salt, key)
    state ← InitState()
    state ← ExpandKey(state, salt, key)
    repeat  (2^cost)
        state ← ExpandKey(state, 0, salt)
        state ← ExpandKey(state, 0, key)
    return state
```

```
state ← ExpandKey(state, 0, salt)
```

Usa come salt il testo “0” e come chiave il salt (128 bits).

```
state ← ExpandKey(state, 0, key)
```

Usa come salt il testo “0” e come chiave la chiave primaria.

Fonte: [12]

bcrypt(cost, salt, pwd)

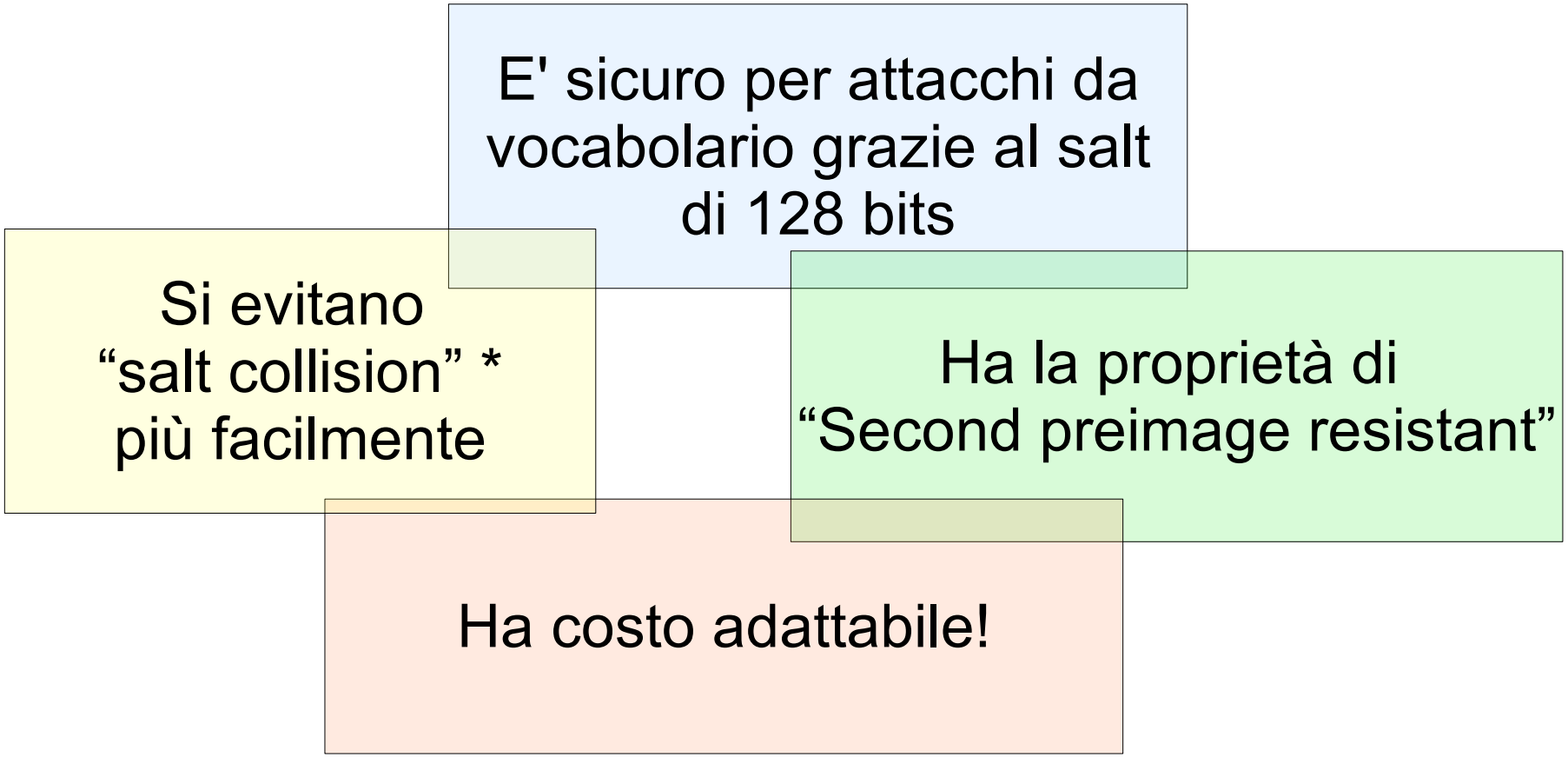
- Salt di 128 bits;
- Cifra un valore magico di 192 bits;
- Sfrutta il “costoso” metodo di generazione delle sottochiavi di `EksBlowfishSetup`.

```
bcrypt(cost, salt, pwd)
    state ← EksBlowfishSetup(cost, salt, key)
    ctext ← “OrpheanBeholderScryDoubt”
    repeat (64)
        ctext ← EncryptECB(state, ctext)
    return Concatenate(cost, salt, ctext)
```

eksblowfish in ECB Mode: ogni blocco di 64 bits viene cifrato con la stessa chiave).

Fonte: [12]

bcrypt: sicurezza



Valori usati di cost (su passwd.conf) alla pubblicazione dell'articolo sono:
6 utente normale; 8 superuser.

Fonte: [12]

* = salt collision: quando due password vengono cifrate con lo stesso salt.

Bibliografia

[1] Bruce Schneier, Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish), Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994

[2] Bruce Schneier, http://www.schneier.com

[3] B. Schneier, D. Whiting, Fast Software Encryption: Designing Encryption Algorithms for Optimal Software Speed on the Intel Pentium Processor, Fast Software Encryption, Fourth International Workshop Proceedings (January 1997), Springer-Verlag, 1997

[4] Douglas R. Stinson, Cryptography. Theory and Practice. Third Edition, CRC, 2006

[5] Claude E. Shannon, Communication Theory of Secrecy Systems, Bell System Technical Journal vol 28-4, 1949

[6] Wikipedia, http://it.wikipedia.org/wiki/Rete_di_Feistel (dicembre 2008)

[7] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC, 1997

[8] Serge Vaudenay, On the Weak Keys of Blowfish, Laboratorie d'Informatique de l'Ecole Normale Sup□ieure, November 1995

[9] Bruce Schneier, The Blowfish Encryption Algorithm -- One Year Later, Dr. Dobb's Journal, September 1995

[10] Dieter Schmidt, On the Key Schedule of Blowfish, Technical report, February 2005

[11] Wikipedia, http://it.wikipedia.org/wiki/Blowfish

[12] Niels Provos, David Mazi□es, A Future-Adaptable Password Scheme, Proceedings of the Annual USENIX Technical Conference, 1999

Conclusioni...

Blowfish è un cifrario molto interessante, soprattutto per come genera le sottochiavi utilizzate poi per la cifratura dei dati.

La versione di Blowfish completa (a 16 rounds) è considerata sicura ([2], [9]).

- Non ne esistono crittoanalisi conosciute.
- Le debolezze elencate non creano dunque problemi reali al cifrario ([9], [10]).

Blowfish è molto usato nel mondo reale, grazie ai suoi aspetti di sicurezza e del fatto del suo codice libero.

Il sito di Schneier elenca diverse applicazioni che usano questo cifrario ([2]).

Si è poi visto bcrypt ([12]), cioè l'implementazione in OpenBSD della funzione di hashing delle passwords per la loro protezione nel file delle password.

- Tale algoritmo permette di mantenere sicuro il sistema per gli anni a venire grazie alla “parametrizzazione” del costo della computazione e di un salt molto grande.