Introduzione allo sviluppo di applicazioni sicure con l'utilizzo di tecniche crittografiche

di Enrico Zimuel (enrico@zimuel.it)

29 Novembre - LinuxDay 2003 – Teramo
Istituto Zooprofilattico Sperimentale dell'Abruzzo e del Molise

Note sul copyright (copyfree):

Questa presentazione può essere utilizzata liberamente
a patto di citare la fonte e non stravolgerne il contenuto.

Questa presentazione è stata creata con OpenOffice 1.0
www.openoffice.org

Questa presentazione è disponibile all'url http://www.zimuel.it/conferenze.htm

Sommario

- Introduzione all'uso degli algoritmi crittografici
- La crittografia è sinonimo di sicurezza?
- Il problema delle password
- Esempi di schemi di autenticazione
- MySQL e crittografia
- Php e crittografia
- Esempi d'utilizzo delle funzioni hash() e sha1()
- Le librerie mcrypt ed mhash

Introduzione all'uso degli algoritmi crittografici

- Perchè utilizzare la crittografia?

- La crittografia può essere utilizzata all'interno di applicazioni software per realizzare le seguenti operazioni: riservatezza, autenticazione, integrità, non ripudio.

- Un programmatore come può utilizzare efficacemente la crittografia all'interno delle proprie applicazioni?

- E' necessario affidarsi a librerie software standard, collaudate e soprattutto open source.

- Perchè reinventare la ruota? Esistono molte librerie valide per la maggior parte dei linguaggi di programmazione. Non cimentatevi nella costruzione di algoritmi di cifratura personalizzati, utilizzate quelli già esistenti.

La crittografia è sinonimo di sicurezza?

- No! La crittografia è uno strumento necessario per la sicurezza ma non sufficiente. Sicurezza e crittografia sono due cose diverse.

- La maggior parte dei problemi legati al mondo della sicurezza informatica deriva da problemi di natura umana/organizzativa.

- Uno stupendo algoritmo di crittografia non può nulla contro una pessima programmazione, un sistema operativo scadente o una password inappropriata.

- “La sicurezza di un sistema informatico non è un prodotto ma un processo” Bruce Schneier

La crittografia open source

- Uno dei principi fondamentali della crittografia è il principio di Kerckhoffs: *"La sicurezza di un sistema crittografico è basata **esclusivamente** sulla conoscenza della chiave, in pratica si presuppone noto a priori l'algoritmo di cifratura e decifrazione."*

- Conoscenza algoritmo = libera distribuzione codici sorgenti = standard aperti = uno dei principi fondamentali dell'open source!

- I sistemi crittografici proprietari chiusi **non possono essere per definizione sicuri**.

- Come fa un sistema chiuso ad essere considerato sicuro se non si conosco, nel dettaglio, i principi di funzionamento?

- "*Se un sistema è veramente sicuro, lo è anche quando i dettagli divengono pubblici*" Bruce Schneier

Il problema della password o della chiave

- Ora che avete scelto il vostro sistema di crittografia “aperto”, dovete utilizzarlo... quasi tutti i sistemi crittografici si basano sull'utilizzo di una password... quale password scegliere?

- La data di nascita di mio figlio? Il nome del mio cane? Il PIN del mio Bancomat che è scritto su di un foglietto conservato nel mio portafoglio...

- Il concetto stesso di password si basa su di un ossimoro: la password deve infatti essere una stringa di caratteri casuali, facile da ricordare.

- Ma se deve essere facile da ricordare come può essere casuale?

Lunghezza delle chiave e sicurezza

- Chiavi più lunghe = chiavi più sicure? In teoria si, in pratica no.

- Le chiavi vengono generate quasi sempre attraverso delle password di autenticazione scelte dall'utente (ad esempio la *pass phrase* del Pgp/GnuPg).

- Quasi sempre le password di autenticazione sono frasi di senso compiuto o frasi casuali di piccole dimensioni, ad esempio di 10 caratteri, è difficile ricordare a memoria più di 10 caratteri casuali.

- Questo limite umano fa diminuire notevolmente lo spazio delle chiavi ossia l'insieme di tutte le possibili permutazioni di una chiave di n bit.

- Tramite dei semplici programmi di cracking è possibile effettuare con successo un attacco di forza bruta (brute-forcing).

Lunghezza delle chiave e sicurezza

- Ad esempio se la password di un utente è una parola di senso compiuto tramite un attacco di forza bruta basato su un dizionario è possibile violare un sistema in pochi secondi.

- Alcuni test effettuati nel 2000 con un famoso programma di cracking, L0phtcrack, hanno dimostrato che il 90% delle password possono essere determinate in meno di un giorno e circa il 20% nell'arco di pochi minuti.

- Se in un sistema che contiene 1.000 account 999 utilizzano password incredibilmente complicate, L0phtcrack riesce a entrare nel sistema scoprendo l'unica password debole.

- Le password di autenticazione sono dunque insicure proprio perchè subentra nel sistema di sicurezza il fattore umano, le password devono essere ricordate dagli utenti.

Consigli sull'utilizzo delle librerie crittografiche

- Mai memorizzare le password/chiavi degli algoritmi di cifratura in chiaro!

- Mai utilizzare come chiavi degli algoritmi di cifratura le password inserite dagli utenti.

- Utilizzare le funzioni hash (MD5, SHA1, etc) per la memorizzazione sicura delle password.

- Prima di cifrare un messaggio, soprattutto se di dimensioni elevate, è consigliabile comprimerlo per aumentarne l'entropia (ad esempio per i file si può utilizzare la compressione gzip prima dell'operazione di encryption).

- Cercate di non utilizzare sempre la stessa chiave di cifratura. Per le password evitate di utilizzare password senza scadenza.

Quali algoritmi crittografici scegliere?

- Non esiste il migliore algoritmo di cifratura, a seconda delle esigenze del progetto si utilizzano algoritmi differenti.

- Allo stato attuale alcuni degli algoritmi crittografici ritenuti sicuri sono:

  - Funzioni hash: MD5, SHA-1, SHA-256, SHA-384, SHA-512.

  - Cifratura simmetrica: Blowfish, Twofish, AES (Rijndael), 3DES, RC5, CAST-128.

  - Cifratura streaming: RC4, SEAL.

  - Cifratura asimmetrica: RSA, ElGamal, DSA.

Quali librerie crittografiche scegliere?

- Esistono moltissime librerie crittografiche per la maggior parte dei linguaggi di programmazione.

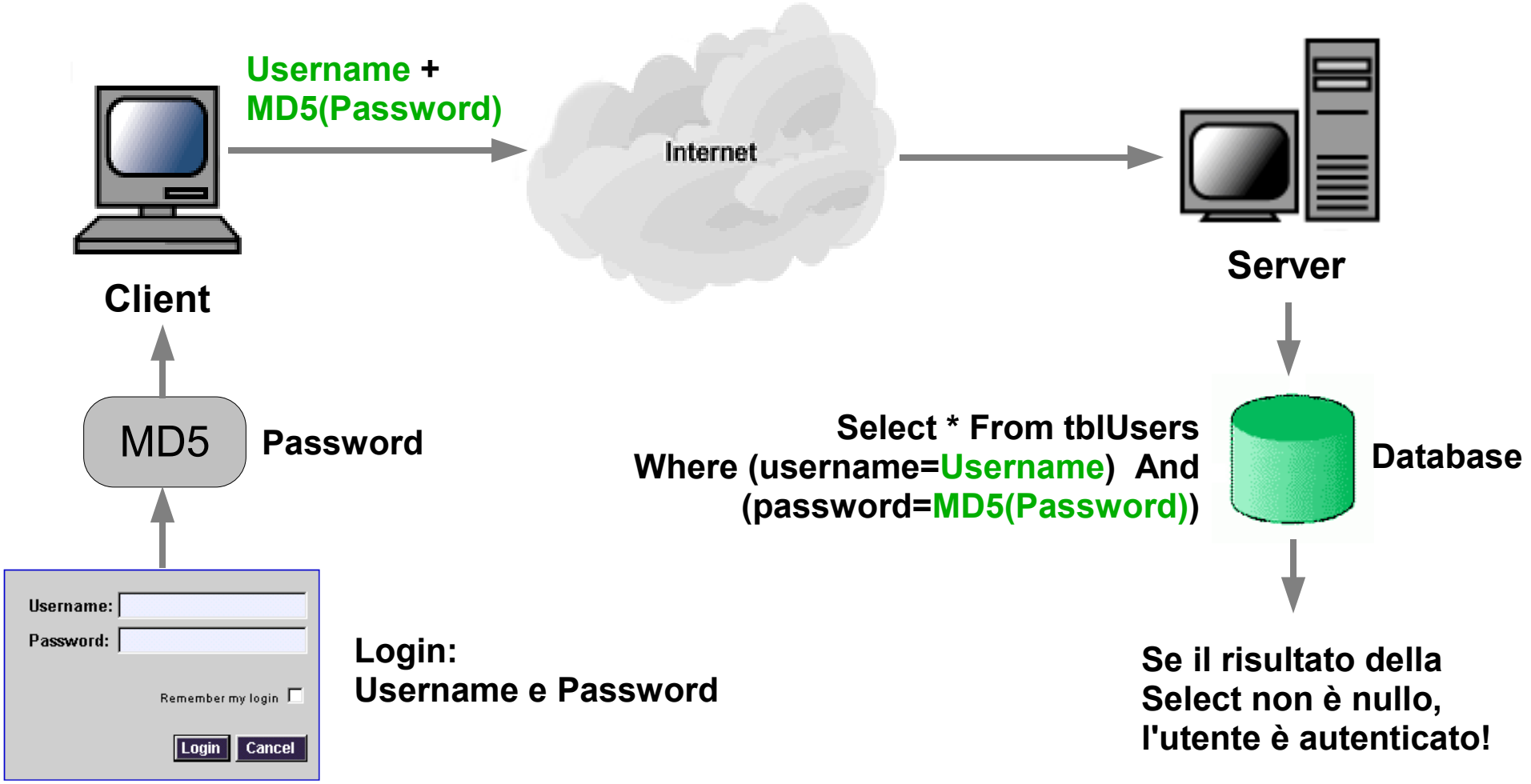
- C,C++: Crypto++, libgcrypt, Botan, OpenCL, BeeCrypt, Ecc, Nettle, OpenSSL.

- PHP: funzioni crypt(), md5(), sha1() e librerie mcrypt, mhash, OpenSSL, CrackLib.

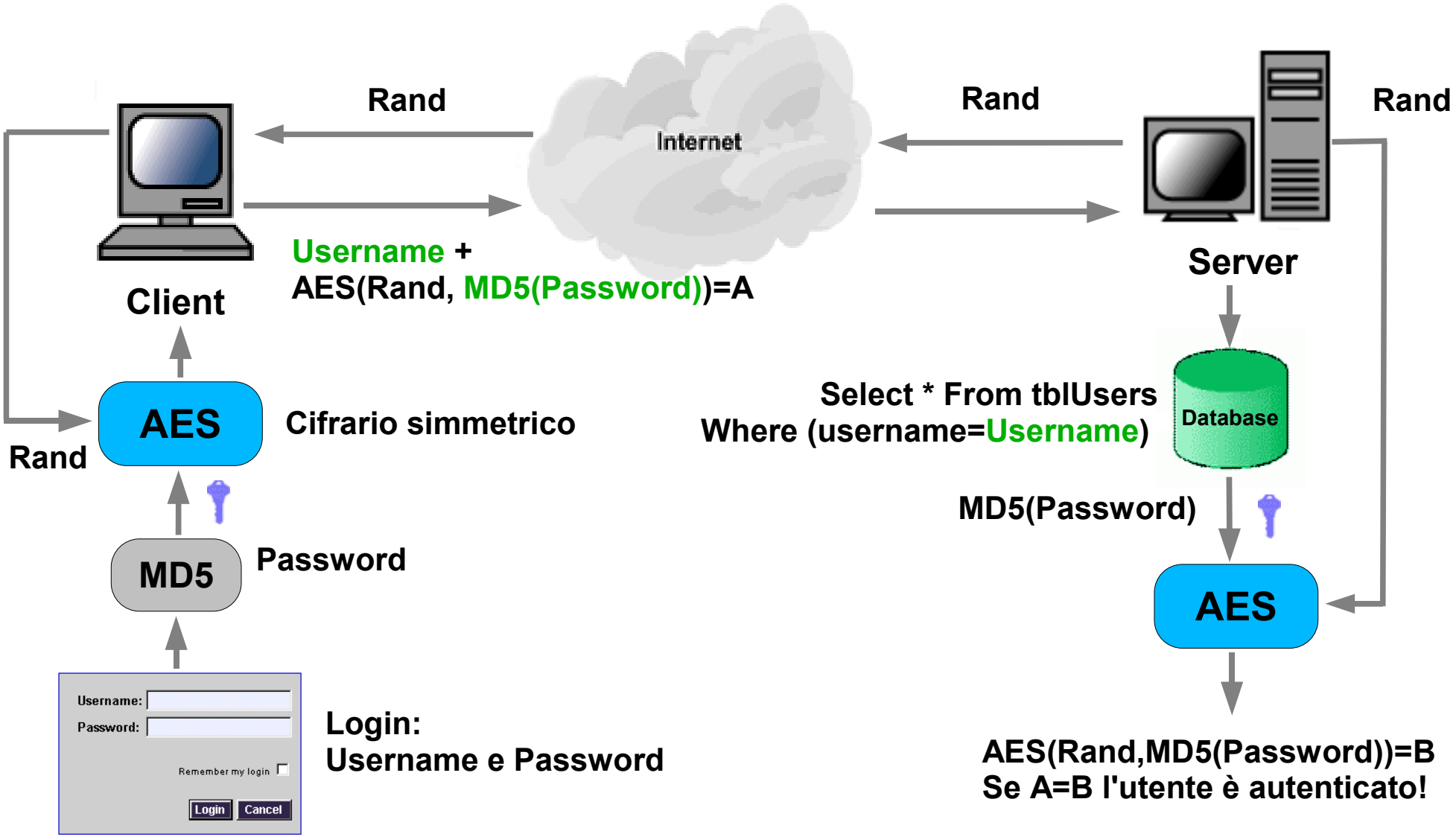
- PERL: CPAN Crypt, Cryptix Perl,

- Java: Java Cryptography Extension (JCE), Cryptix JCE.

Autenticazione tramite calcolo dell'hash MD5 della password



Autenticazione tramite Challenge and response



Nota: la chiave degli algoritmi simmetrici AES è l'MD5(Password) indicata con l'icona

MySQL e crittografia

- Il database MySQL ha al suo interno una serie di istruzioni crittografiche in SQL molto potenti.

- Alcune funzioni crittografiche presenti sono: MD5(str), SHA1(string), ENCRYPT(str[,salt]), AES_ENCRYPT(str,key_string), AES_DECRYPT(str,key_string).

- Ad esempio è possibile cifrare il contenuto di un record:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
```

N.B. Il campo con l'output dell'AES_ECNRYPT deve essere di tipo Text o Blob ;

- O calcolare l'MD5 di una stringa:

```
mysql> SELECT MD5("testing");
       -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

PHP e crittografia

- Le funzioni crittografiche presenti nella distribuzione standard del PHP sono le funzioni stringa crypt(), md5() e sha1().

- Queste funzioni consentono di generare solo degli hash code e quindi non possono essere utilizzate per le operazioni di cifratura dei dati.

- Anche la funzione crypt(), a dispetto del nome, è una funzione hash one-way. E' stata chiamata in questo modo per motivi di compatibilità con la funzione crypt di Unix utilizzata per la memorizzazione sicura delle password degli utenti.

- Le altre due funzioni md5() ed sha1() implementano rispettivamente le funzioni hash MD5 ed SHA1.

La funzione hash crypt()

- Il prototipo della funzione crypt() è:

string crypt (string str [, string salt])

dove str è la stringa in input della funzione hash e salt è un parametro stringa opzionale che viene concatenato con la stringa str per il calcolo dell'hash. In pratica questo parametro viene utilizzato per aumentare la sicurezza dell'hash contro attacchi basati su dizionari. Se questo parametro non è specificato il Php genera automaticamente un salt casuale ad ogni esecuzione della funzione.

- La lunghezza del salt dipende dall'algoritmo utilizzato per il calcolo dell'hash che dipende a sua volta dal sistema operativo utilizzato. Ad esempio su un sistema Red Hat 9 Linux con Kernel 2.4.20 il salt è di 12 caratteri poichè l'algoritmo utilizzato per l'hash è l'MD5.

La funzione hash crypt()

- Esempio d'utilizzo della funzione crypt():

```
<?php
   $password= "test";
   $digest= crypt($password);

   echo "Hash('".$password."')= ".$digest;
?>
```

- Ogni volta che si esegue questo script il valore di $digest cambia, ad esempio:

Hash('test')= 1dLygajpl$zJ3E92oyju7vAYZ7MuQb8.
Hash('test')= 1Spznq1WT$udgFuVOf8Eyg8Ca815ltX1
Hash('test')= $1$9oJmpneM$.c/f2UwF0zJ5GLMGMlCef0

I caratteri in rosso sono i salt generati automaticamente da Php. La stringa iniziale 1 identifica l'MD5 (costante CRYPT_MD5).

La funzione hash crypt()

- Per eseguire il confronto con un valore hash memorizzato è necessario riapplicare lo stesso salt.
- Esempio di autenticazione della password tramite hash e valore inviato tramite GET http://server/esempio.php?password=test

```
<?php
   $password="test";
   $digest= crypt($password);
   $newdigest= crypt($_GET['password'],$digest);

   if ($newdigest==$digest) {
      echo "Utente autenticato! :-)";
   } else {
      echo "Utente non autenticato! :-(";
   }
?>
```

- Al posto del salt $digest si può anche sostituire substr($digest,0,12) oppure substr($digest,0,11).

Le funzioni hash md5() e sha1()

- L'utilizzo delle funzioni md5() e sha1() è più immediato rispetto a crypt() poichè è sufficiente specificare solo la stringa di input della funzione hash senza il salt.
- Il prototipo delle due funzioni:

```
string md5  ( string str [, bool raw_output] )
string sha1 ( string str [, bool raw_output] )
```

la funzione md5() è disponibile a partire dalla versione 3 del Php mentre l'sha1() a partire dalla versione 4.3.0.

- La funzione md5() restituisce una stringa esadecimale di 32 caratteri. Se il parametro opzionale raw_output è impostato a TRUE la funzione restituisce una valore binario di lunghezza 16.
- La funzione sha1() restituisce una stringa esadecimale di 40 caratteri, con raw_output==TRUE un valore binario di 20 byte.

Le funzioni hash md5() e sha1()

- Esempio di sistema di autenticazione tra client e server con l'utilizzo della funzione hash md5() ed un database MySQL.

```
<?php
$host = "localhost"; $user = "test"; $pswd = ""; $db = "utenti";

// Imposto l'autorizzazione a false
$autorizzazione=0;

// Verifico che l'utente abbia introdotto la username e la
password
if (isset($_SERVER['PHP_AUTH_USER']) &&
isset($_SERVER['PHP_AUTH_PW'])) {

   mysql_pconnect($host, $user, $pswd) or die("Non riesco a
collegarmi al server MySQL!");
   mysql_select_db($db) or die("Non riesco a collegarmi al db!");

?>
```

Segue...

Le funzioni hash md5() e sha1()

```
<?php
    // Calcolo l'hash del valore inserito dall'utente
    $digest = md5($_SERVER['PHP_AUTH_PW']);

    $query = "SELECT username FROM utenti WHERE username = '" .
$_SERVER['PHP_AUTH_USER'] ." AND password = '$digest'";

   if (mysql_numrows(mysql_query($query)) == 1) {
      $autorizzazione=1;
   }
}

if ($autorizzazione==0) {
  header('WWW-Authenticate: Basic realm="Private"');
  header('HTTP/1.0 401 Unauthorized');
  print "Non sei autorizzato ad accedere al sito.";
  exit;
} else {
  print "Autorizzazione concessa!";
}
?>
```

Le funzioni hash md5() e sha1()

- Ovviamente il database MySQL degli utenti deve essere preventivamente generato con il calcolo dell'MD5 delle password.

- Il vantaggio principale di una soluzione del genere consiste nella memorizzazione sicura del database delle password. In pratica se il database degli utenti viene violato le password rimangono comunque al sicuro proprio perchè non è possibile ricavarne il valore in chiaro partendo dall'hash.

- Volendo migliorare lo script precedente non si dovrebbe inviare in chiaro la password di autenticazione tra il client ed il server. Questo può essere ottenuto tramite un collegamento SSL nel quale si cifra tutto il traffico HTTP tra client e server oppure attraverso l'utilizzo di una funzione hash client, ad esempio in javascript, che calcoli l'hash prima di inviare il valore tramite un submit in POST. Esiste anche un modulo di autenticazione per Apache (mod_auth_digest) ed una libreria PHPlib...

Le funzioni hash md5_file() e sha1_file()

- Esistono due funzioni per il calcolo dell'hash MD5 e SHA1 applicabili su interi file.
- Il prototipo di queste due funzioni è identico a quello dell'md5() e dell'sha1():

```
string md5_file  ( string filename [, bool raw_output] )
string sha1_file ( string filename [, bool raw_output] )
```

in questo caso il parametro filename indica il nome del file in input alla funzione hash.

- Queste due funzioni, disponibili a partire dalla versione 4.2.0 di Php, sono molto utili perchè consentono di risolvere il problema dell'integrità degli script Php ed in generale dei file memorizzati su di un server.
- Calcolando l'hash di un sorgente Php è possibile stabilire, a run-time, se lo script è stato modificato o meno da un eventuale intruso...

La libreria mcrypt

- La libreria mcrypt non è presente nell'installazione di default di Php.

- Essa è solo un modulo d'interfaccia alla libreria GPL libmcrypt disponibile all'indirizzo mcrypt.hellug.gr.

- La libreria libmcrypt supporta i seguenti algoritmi di cifratura: BLOWFISH, TWOFISH, DES, TripleDES, 3-WAY, SAFER-sk64, SAFER-sk128, SAFER+, LOKI97, GOST, RC2, RC6, MARS, IDEA, RIJNDAEL-128 (AES), RIJNDAEL-192, RIJNDAEL-256, SERPENT, CAST-128 (conosciuto come CAST5), CAST-256, ARCFOUR e WAKE.

- Attualmente la versione stabile è la 2.5.7.

L'installazione della libreria mcrypt

- Per installare la libreria libmcrypt-x.x.tar.gz è necessario compilare i sorgenti con la seguente procedura:

```
gunzip libmcrypt-x.x.tar.gz
tar -xvf libmcrypt-x.x.tar.gz
./configure --disable-posix-threads
make
make install
```

- Una volta installata la libreria è possibile ricompilare il Php con la direttiva -with-mcrypt utilizzando la seguente procedura:

```
cd [dir PHP]
./configure -with-mcrypt=[dir libmcrypt]
make
make install
```

[dir PHP] e [dir libmcrypt] sono le directory di installazione del PHP e della libreria libmcrypt rispettivamente.

La libreria mcrypt

- La libreria mcrypt ha a disposizione le seguenti funzioni:

mcrypt_cbc -- Encrypt/decrypt data in CBC mode
•mcrypt_cfb -- Encrypt/decrypt data in CFB mode
•mcrypt_create_iv -- Create an initialization vector (IV) from a random source
•mcrypt_decrypt -- Decrypts crypttext with given parameters
•mcrypt_ecb -- Encrypt/decrypt data in ECB mode
•mcrypt_enc_get_algorithms_name -- Returns the name of the opened algorithm
•mcrypt_enc_get_block_size -- Returns the blocksize of the opened algorithm
•mcrypt_enc_get_iv_size -- Returns the size of the IV of the opened algorithm
•mcrypt_enc_get_key_size -- Returns the maximum supported keysize of the opened mode
•mcrypt_enc_get_modes_name -- Returns the name of the opened mode
•mcrypt_enc_get_supported_key_sizes -- Returns an array with the supported keysizes of the opened algorithm
•mcrypt_enc_is_block_algorithm_mode -- Checks whether the encryption of the opened mode works on blocks
•mcrypt_enc_is_block_algorithm -- Checks whether the algorithm of the opened mode is a block algorithm
•mcrypt_enc_is_block_mode -- Checks whether the opened mode outputs blocks
•mcrypt_enc_self_test -- This function runs a self test on the opened module
•mcrypt_encrypt -- Encrypts plaintext with given parameters
•mcrypt_generic_deinit -- This function deinitializes an encryption module
•mcrypt_generic_end -- This function terminates encryption
•mcrypt_generic_init -- This function initializes all buffers needed for encryption

Segue...

La libreria mcrypt

- mcrypt_generic -- This function encrypts data
- mcrypt_get_block_size -- Get the block size of the specified cipher
- mcrypt_get_cipher_name -- Get the name of the specified cipher
- mcrypt_get_iv_size -- Returns the size of the IV belonging to a specific cipher/mode combination
- mcrypt_get_key_size -- Get the key size of the specified cipher
- mcrypt_list_algorithms -- Get an array of all supported ciphers
- mcrypt_list_modes -- Get an array of all supported modes
- mcrypt_module_close -- Close the mcrypt module
- mcrypt_module_get_algo_block_size -- Returns the blocksize of the specified algorithm
- mcrypt_module_get_algo_key_size -- Returns the maximum supported keysize of the opened mode
- mcrypt_module_get_supported_key_sizes -- Returns an array with the supported keysizes of the opened algorithm
- mcrypt_module_is_block_algorithm_mode -- This function returns if the the specified module is a block algorithm or not
- mcrypt_module_is_block_algorithm -- This function checks whether the specified algorithm is a block algorithm
- mcrypt_module_is_block_mode -- This function returns if the the specified mode outputs blocks or not
- mcrypt_module_open -- Opens the module of the algorithm and the mode to be used
- mcrypt_module_self_test -- This function runs a self test on the specified module
- mcrypt_ofb -- Encrypt/decrypt data in OFB mode

mdecrypt_generic -- Decrypt data

La libreria mcrypt

- Le funzioni principali della libreria mcrypt() sono le funzioni di cifratura e decifrazione mcrypt_encrypt() e mcrypt_decrypt():

string **mcrypt_encrypt** (string cipher, string key, string data, string mode [, string iv])
string **mcrypt_decrypt** (string cipher, string key, string data, string mode [, string iv])

dove:
- *cipher* è il tipo di algoritmo, specificato tramite le costanti MCRYPT_ciphername (ad esempio MCRYPT_BLOWFISH);
- *key* è la chiave di cifratura, è consigliabile non utilizzare una stringa Ascii ma l'ouput della funzione mhash_keygen_s2k (vedremo come);
- *data* è la stringa contenente i dati da cifrare;
- mode è la modalità di cifratura dell'algoritmo utilizzato (ECB,CBC,CFB,OFB,NOFB,STREAM);
- *iv* è un parametro opzionale utilizzato sono nelle modalità di cifratura CBC, CFB, OFB ed in qualche algoritmo di STREAM.

Le modalità di cifratura simmetriche

- In crittografia gli algoritmi di cifratura simmetrici si dividono nelle due seguenti categorie: a *blocchi* (Block Cipher) e a *sequenza* (Stream Cipher).

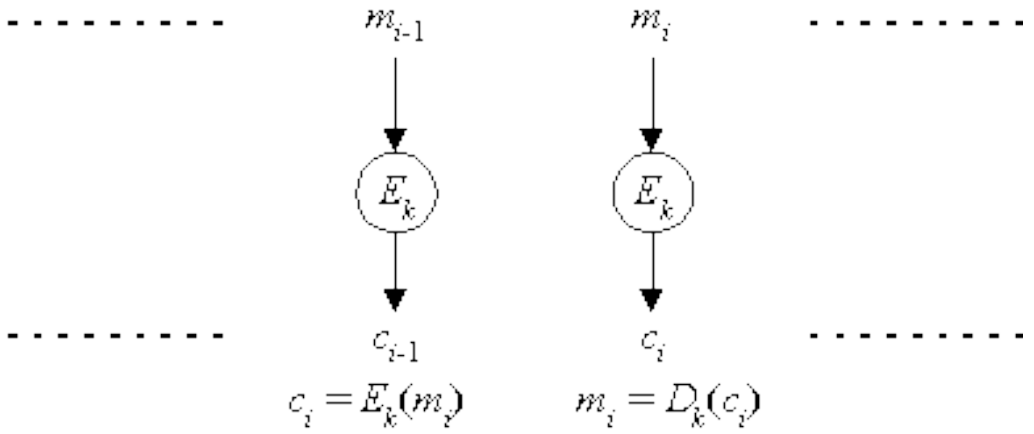
- Nei cifrari a *blocchi* il testo in chiaro viene suddiviso in blocchi di n bit (di solito 64) ed ogni blocco viene cifrato tramite l'algoritmo di cifratura. Nei cifrari a *sequenza* il testo in chiaro viene cifrato un bit o byte per volta come in una catena di montaggio.

- Con un cifrario a *blocchi*, tenendo costante la chiave k di cifratura, ad ogni blocco di testo viene sempre associato lo stesso crittogramma. Invece in un cifrario a *sequenza* il testo in chiaro viene cifrato ogni volta in modo differente.

- A loro volta sia cifrari a *blocchi* che quelli a *sequenza* si differenziano ulteriormente per le loro modalità operative.

Tipologie di cifrari a blocchi: ECB

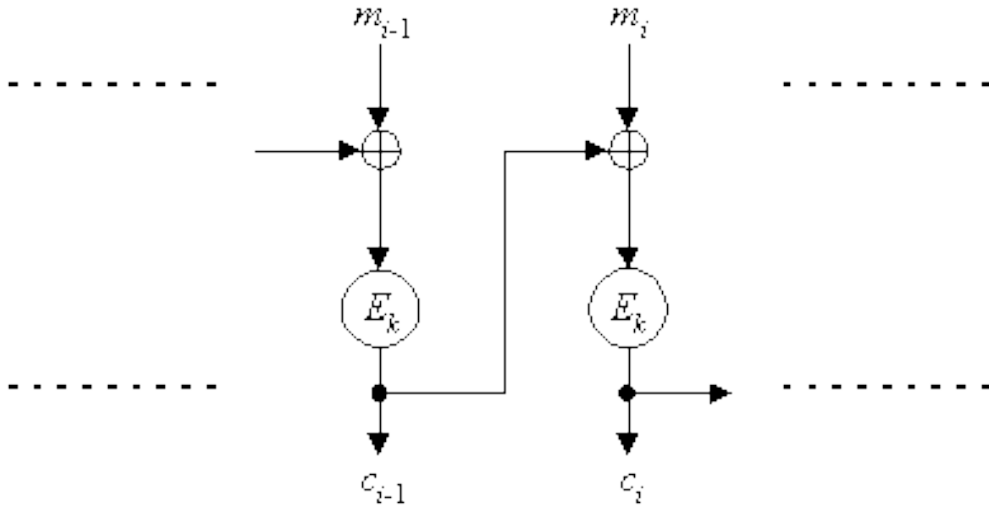
- I cifrari a blocchi possono operare nelle seguenti modalità: ECB (Electronic CodeBook), CBC (Cipher Block Chaining), CFB (Cipher FeedBack), OFB (Output FeedBack).
- La modalità ECB (Electronic CodeBook) è la più semplice ed intuitiva, infatti in essa ogni blocco viene cifrato con la stessa chiave k. Il crittogramma risulta quindi formato da una sequenza di blocchi cifrati ognuno allo stesso modo.



Electronic Code Book

Tipologie di cifrari a blocchi: CBC

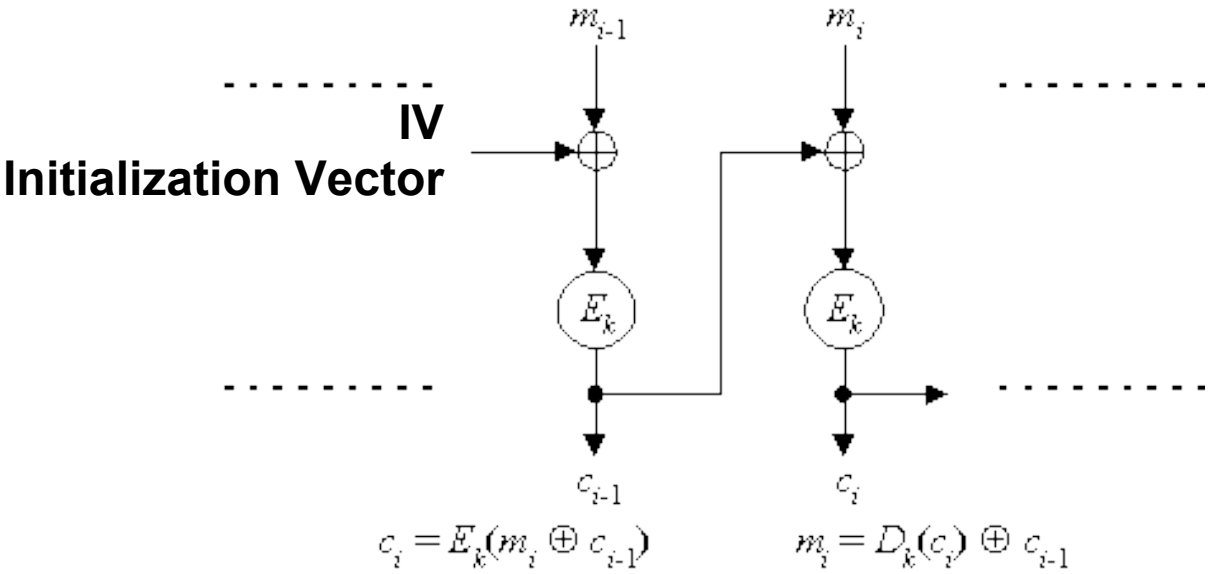
- Nella modalità CBC (Cipher Block Chaining) i blocchi vengono cifrati utilizzando il crittogramma del blocco precedente. Si esegue un'operazione di XOR tra il crittogramma precedente c_{i-1} ed il blocco in chiaro m_i ed il risultato viene cifrato con l'algoritmo E_k ottenendo il crittogramma c_i. In simboli: $c_i = E_k (m_i \oplus c_{i-1})$



Cipher Block Chaining

Tipologie di cifrari a blocchi: CBC

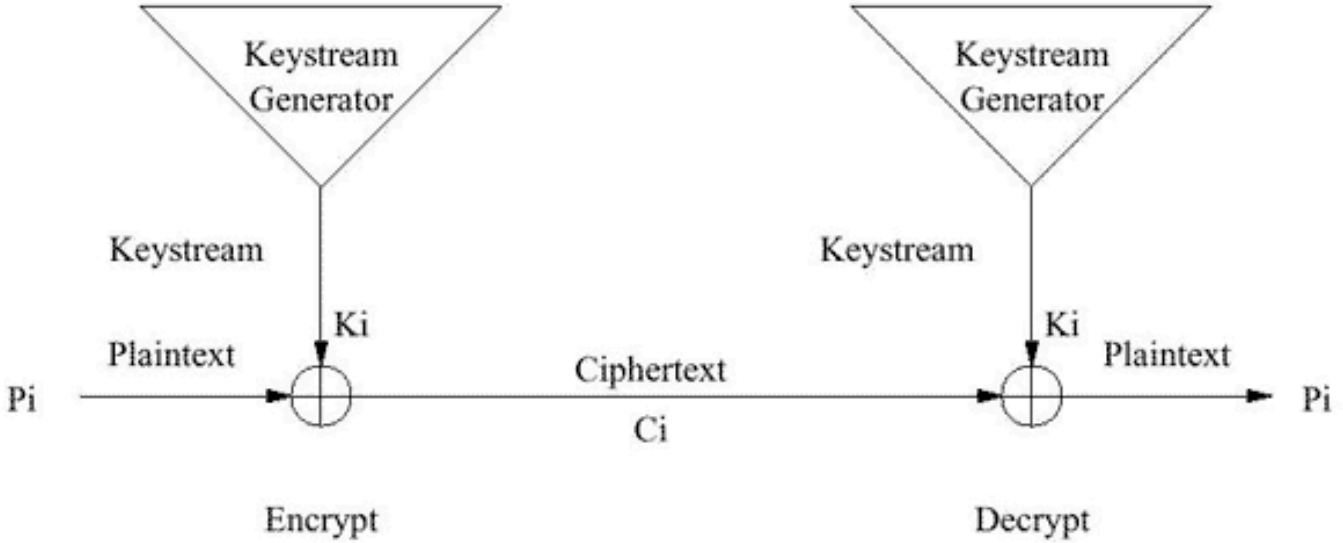
- Per migliorare la sicurezza dei cifrari a blocchi CBC si utilizza un *vettore di inizializzazione* (Initialization Vector, IV), il più delle volte generato casualmente, come input per il primo blocco. Tale vettore consente di variare il crittogramma a parità di testo in chiaro m e chiave k.
- Il vettore di inizializzazione può rimanere tranquillamente in chiaro e venire così trasmesso con il crittogramma.



Cipher Block Chaining

I cifrari a sequenza (stream cipher)

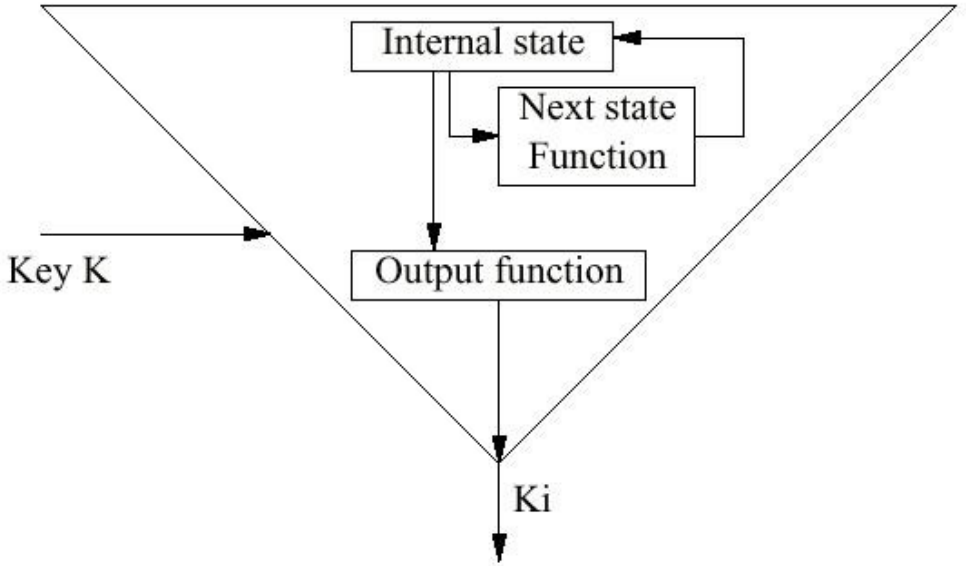
- I cifrari a sequenza (stream cipher) convertono il testo in chiaro nel crittogramma un bit o byte per volta. L'implementazione più semplice di un cifrario a sequenza è quella descritta dal seguente schema:



- La funzione di cifratura e di decifrazione è costituita semplicemente dall'operatore XOR (⊕). Ogni bit o byte del crittogramma C_i viene ottenuto con l'operazione di XOR tra il bit o byte del testo in chiaro P_i e il bit o byte del keystream K_i.

I cifrari a sequenza (stream cipher)

- Il Keystream è un generatore di numeri che accetta come input la chiave k di cifratura.



- La sicurezza dei cifrari a sequenza dipende interamente dal keystream. Una delle caratteristiche fondamentali di un buon keystream è quella di generare numeri i più casuali possibili (pseudo-casuali).

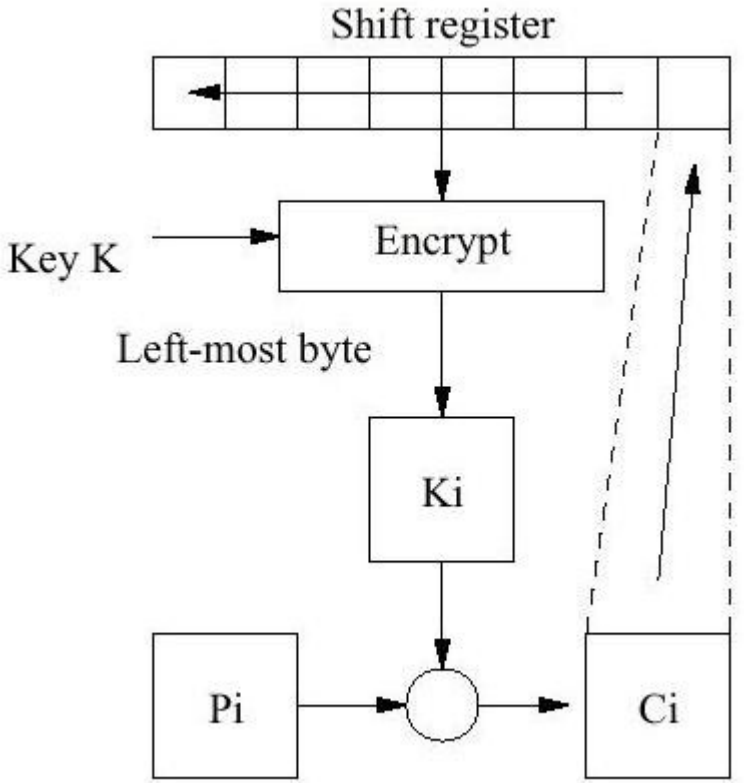
Tipologie di cifrari a blocchi: CFB

- Nei cifrari a blocchi CFB (Cipher FeedBack) si utilizzano le tecniche dei cifrari a sequenza combinate con la tecnica CBC. Nella modalità CBC i blocchi vengono cifrati utilizzando l'ultimo crittogramma generato con un'operazione di XOR. In questo modo l'operazione di cifratura ha inizio solo quando il blocco in chiaro viene ricevuto interamente. In alcuni ambiti applicativi questa modalità può essere un limite in termini di interattività (si pensi ad esempio agli emulatori di terminale).

- Nella modalità CFB i blocchi vengono cifrati suddividendoli in unità più piccole (di n bit) utilizzando un registro a scorrimento lineare. Queste unità vengono cifrate attraverso l'utilizzo di un CBC con la chiave k di cifratura. Gli n-bit più significativi vengono combinati con gli n-bit del testo in chiaro tramite un'operazione di XOR. Il crittogramma generato viene utilizzato per le operazioni successive di cifratura tramite il reinserimento nel registro a scorrimento lineare.

Tipologie di cifrari a blocchi: CFB

- Sequenza delle operazioni di cifratura per un 8-bit CFB:
  1. i=1, lo shift register viene riempito con il vettore di inizializzazione IV (come nel CBC);
  2. Il registro viene cifrato con la chiave K;
  3. K_i= 8 bit più significativi del crittogramma;
  4. P_i= 8 bit del testo in chiaro;
  5. $C_i = P_i \oplus K_i$ (stream cipher);
  6. Lo shift register viene aggiornato con i valori del crittogramma C_i.
  7. Si passa allo step successivo i+1;

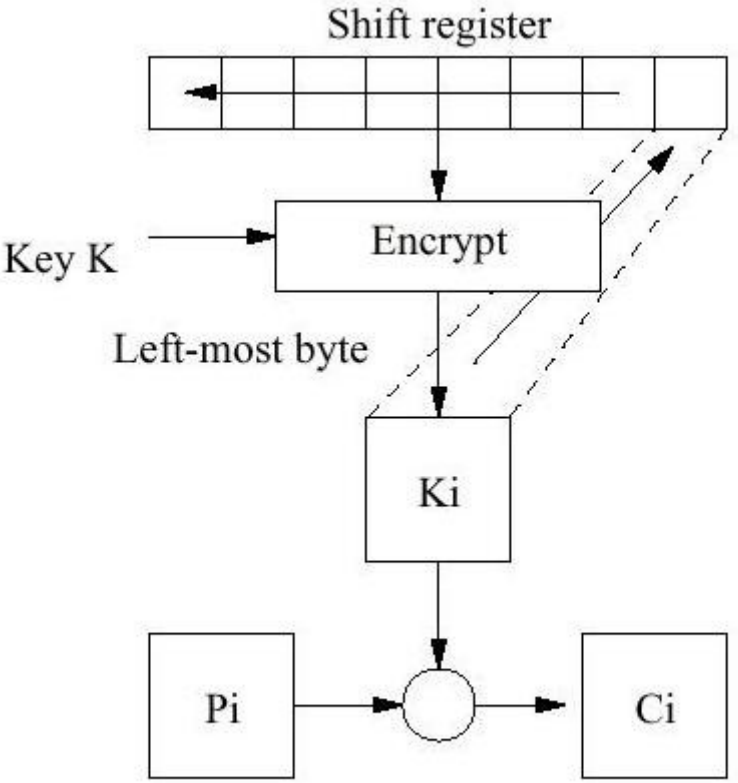


Tipologie di cifrari a blocchi: OFB

- Il metodo OFB (Output FeedBack) è un modo di eseguire un cifrario a blocchi come uno cifrario a sequenza sincrono (synchronus stream cipher).

- E' simile al CFB solo che in questa nuova modalità il registro (shift register) viene rigenerato dagli n-bit più significativi del crittogramma generato dalla chiave K, K_i.

- Sia in cifratura che in decifrazione la tecnica OFB utilizza l'algoritmo di Encrypt nella stessa modalità di cifratura. Questa modalità viene spesso chiamata **internal feedback**.



Consigli sulla scelta delle tipologie di cifrari a blocchi

- In generale valgono le seguenti considerazioni:
  - ECB (Eletronic CodeBook) è consigliabile quando i dati da cifrare sono casuali (random) e di dimensioni limitate;
  - CBC (Cipher Block Chaining) è la modalità adatta per la cifratura dei file;
  - CFB (Cipher Feedback) è la modalità adatta per la cifratura di sequenze di byte dove è necessario cifrare ogni singolo byte (stream).
  - OFB (Output Feedback) è paragonabile al CFB e può essere utilizzato in quelle applicazioni dove la propagazione dell'errore è accettabile. Non è particolarmente sicuro! Ne sconsiglio l'utilizzo, nella libreria mcrypt() è meglio usare il metodo NOFB.
  - STREAM, nella libreria mcrypt() del Php esistono solo due algoritmi “puri” di stream l'RC4 ed il WAKE.

Esempio, in PHP, d'utilizzo della libreria mcrypt

- Un esempio d'utilizzo delle funzioni mcrypt_encrypt() e mcrypt_decrypt().

```
<?php
   $iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH,MCRYPT_MODE_CBC);
   $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);

   echo "Dimensione dell'IV: $iv_size<br>";
   echo "IV: ". bin2hex($iv) ."<br><br>";

   $key = "Parola segreta";   $text = "Messaggio di prova.";
   echo "Testo in chiaro: $text<br>"; echo "Chiave (key): $key<br>";

   $crypttext = mcrypt_encrypt(MCRYPT_BLOWFISH, $key, $text,
MCRYPT_MODE_CBC, $iv);
   echo "Crittogramma: ". bin2hex($crypttext). "<br>";

   $decrypted_string = mcrypt_decrypt(MCRYPT_BLOWFISH, $key,
$crypttext, MCRYPT_MODE_CBC, $iv);
   echo "Testo decifrato: $decrypted_string";
?>
```

La libreria mhash

- La libreria mhash non è presente nell'installazione di default di Php.

- Essa è solo un modulo d'interfaccia alla libreria GPL mhash disponibile all'indirizzo mhash.sourceforge.net.

- La libreria mhash supporta i seguenti algoritmi di hash: **SHA1**, **GOST**, **HAVAL256**, **HAVAL224**, **HAVAL192**, **HAVAL160**, **HAVAL128**, **MD5**, **MD4**, **RIPEMD160**, **TIGER**, **TIGER160**, **TIGER128.**

- Attualmente la versione stabile è la 0.8.18. E' disponibile anche una versione per sistemi Windows.

Installazione della libreria mhash

- Per installare la libreria mhash-x.x.tar.gz è necessario compilare i sorgenti con la seguente procedura:

```
gunzip mhash-x.x.tar.gz
tar -xvf mhash-x.x.tar.gz
./configure
make
make install
```

- Una volta installata la libreria è possibile ricompilare il Php con la direttiva --with-mhash utilizzando la seguente procedura:

```
cd [dir PHP]
./configure --with-mhash=[dir mhash]
make
make install
```

[dir PHP] e [dir mhash] sono le directory di installazione del PHP e della libreria mhash rispettivamente.

La libreria mhash

- La libreria mhash() dispone delle seguenti funzioni:

•mhash_count -- Get the highest available hash id
•mhash_get_block_size -- Get the block size of the specified hash
•mhash_get_hash_name -- Get the name of the specified hash
•mhash_keygen_s2k -- Generates a key
mhash -- Compute hash

- La funzione principale è mhash:

string mhash (int hash, string data [, string key])

dove *hash* è il tipo di algoritmo (ad esempio MHASH_MD5), *data* è la stringa per il calcolo dell'hash e *key* è un parametro opzionale che specifica la chiave per il calcolo dell'HMAC.

La libreria mhash

- Esempio d'utilizzo della funzione mhash():

```
<?php
   $input = "esempio di utilizzo di mhash()";
   $hash = mhash (MHASH_MD5, $input);
   echo "Hash: ".bin2hex ($hash)."<br>";
   $hash = mhash (MHASH_MD5, $input, "test");
   print "HMAC: ".bin2hex ($hash)."<br>";
?>
```

- L'output della funzione mhash() è di tipo binario, quindi è necessario convertirlo in esadecimale per poterlo visualizzare correttamente tramite la funzione bin2hex.
- Nel calcolo dell'HMAC (Hash Message Authentication Code) la stringa “test” rappresenta la chiave di cifratura. Quindi $hash alla fine contiene l'MD5 della stringa $input cifrata con la chiave “test”.

La libreria mhash

- La libreria mhash() mette a disposizione una funzione molto utile per generare delle password “sicure” a partire dalle password scelte da un utente.

- Il problema delle password scelte dagli utenti è che esse risultano essere troppo prevedibili. Un semplice attacco basato su un dizionario linguistico, meglio se calibrato sulle abitudini dell'utente, è sufficiente per scovare la password di un utente in pochi secondi. La sicurezza di un sistema non può essere affidata alle password degli utenti!

- “*L'anello più debole di un sistema di sicurezza è il fattore umano*” Kevin D. Mitnick

- La funzione mhash_keygen_s2k della libreria mhash() consente di rimediare, almeno in parte, a questo problema.

La libreria mhash

- Il prototipo della funzione mhash_keygen_s2k() è:

string mhash_keygen_s2k (int hash, string password, string salt, int bytes)

dove *hash* è il tipo di algoritmo hash utilizzato per la generazione della chiave (ad esempio MHASH_MD5), *password* è la password utente che dovrà essere trasformata, *salt* è una stringa pseudo-casuale che viene utilizzata per la generazione, *bytes* è la lunghezza in bytes della password che si desidera generare.

- La lunghezza di *salt* è fissa a 8 bytes, ad ogni generazione di una nuova password il valore di *salt* dovrebbe essere differente in modo da aumentare la sicurezza dell'algoritmo.

- Questa funzione di generazione delle password è basata sull'algoritmo Salted S2K specificato nel documento OpenPGP (RFC 2440).

La funzione mhash_keygen_s2k()

- Ecco un breve esempio sull'utilizzo della funzione mhash_keygen_s2k() per la generazione delle password:

```
<?php
   $password_utente = "test" ;
   // genero il salt casuale tramite md5
   $salt = substr(pack("h*", md5(mt_rand())) , 0, 8);

   // genero una password di 10 bytes

   $password= mhash_keygen_s2k(MHASH_MD5, $clear_pw, $salt, 10);
   echo "Password utente= $password_utente<br>";
   echo "Password generata= $password<br>";
?>
```

- La funzione mhash_keygen_s2k() può essere utilizzata per generare le chiavi di cifratura per gli algoritmi simmetrici della libreria mcrypt()...

- Libri:
  - “Applied Cryptography, second edition” Bruce Schneier (John Wiley & Sons, 1996)
  - “Pratical Cryptography” Niels Ferguson, Bruce Schneier (Wiley Publishing, 2003)
  - “L'arte dell'inganno” Kevin D.Mitnick (Feltrinelli, 2003)

- Siti Internet:
  - Sito ufficiale PHP
  - PHP's Encryption Functionality
  - RFC 1321 - The MD5 Message-Digest Algorithm
  - RFC 3174 - US Secure Hash Algorithm 1 (SHA1)
  - MD5 in Javascript
  - Applying crypt() to User Validation
  - Compiling PHP 4 and Apache 2 from source on Linux OS