Hashing

* Hashing is the term to the process or technique to the string of data and applying mathematical function to it and produce unique string to output which is known as hash.
* Hash is often used for encryption

Keep in Mind about Password

* Never store passwords in plain text
* Always encrypt them

One-way encryption

* One-way means non-reversible, even by us reversible(capable of returning to an original condition)
* One way encryption is a mathematical function that takes a variable-length input string and converts it into a fixed-length binary sequence. Furthermore, one-way encryption, or a one-way hash function, is designed in such a way that it is hard to reverse the process.
* Same inputs + same hashing algorithm = same output
* Actual password gets encrypted, then stored
* Attempted password gets encrypted, compared against stored

Hashing algorithms suitable for passwords

* MD5
* SHA-1
* SHA-2 (SHA-256, SHA-512)
* Whirlpool, Tiger, AES
* Blowfish
* High-level of security
* Public domain, no patents, free to use
* Included with PHP v5.3 and later
* Slow
* Md5($password); // PHP 3
* Sha1($password); // PHP 4.3
* Hash(‘sha1’, $password); // PHP 5.1.2
* Fast, supports many alorithms

- has\_algos();

- crypt ($password, $salt); // PHP 5.3

* Slow, supports six algorithms
* DES, Ext-DES, MD5, SHA-256, SHA-512, Blowfish

Rainbow Table

* Pre-computed tables of password hashes for each hashing algorithm
* A rainbow table is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes. Tables are usually used in recovering a plaintext password up to a certain length consisting of a limited set of characters.
* It does the reverse of a hash function (mapping hashes to plaintexts), but it is /not/ an inverse hash function.

Salt

* Additional data added to the password before encryption
* “Put salt on the {$password}”;
* “e5fasdfasdf96erwer9778dsfasfddasftrttryyugdbhjnmm365”
* Knowing password requires also knowing the salt string
* Rainbow tables could be used, but they would be almost impossibly large

Unique Salt

* Create salt using strings unique to each user
* “Put salt on the {$password} for {$username}”;
* “aeff66faerehfg3933fdsadrew1145fasfaser”
* Knowing the password requires knowing the salt string and the user string
* Rainbow tables are still impossibly large, but now each user’s salt is unique too

Random Salt

* Create random salt using pseudo-random string (e.g. mt\_rand, time)
* “Put salt on the {$password} at “. time() ;
* “b137fdadf9r68dsfdasfa65feiomnhbiqereapp”
* Knowing the password requires knowing the random string
* Rainbow tables are useless, as each user’s hash is almost random, almost unique.

Store Salt in Database

* When using user data for salt and user data could change
* When using random salt
* Just the salt, not the password
* Hash the salt so it will not be plain text
* $salt = md5(uniqid(mt\_rand(), true));
* $format\_and\_salt = $format\_string. $salt;
* $hashed\_password = crypt($password, $format\_and\_salt);

Crypt

* One-way string hashing

Crypt\_Blowfish

* Blowfish hashing with a salt as follows: "$2a$", "$2x$" or "$2y$", a two digit cost parameter, "$", and 22 characters from the alphabet ". /0-9A-Za-z".
* $2y$10$ - $2y$: means use blowfish; 10: means cost parameter how many times will blowfish hash function run.
* Cost parameter range from 04 – 31.
* Blowfish is more secure than md5, sha1, plain text password etc.

New PHP Password Functions

* Password\_hash()
* Password\_hash($password, PASSWORD\_DEFAULT);
* Password\_hash ($password, PASSWORD\_BCRYPT, [‘cost’ => 10);
* Password\_verify()
* Password\_verify($password, $existing\_hash);
* Our version password hashing will be

Password\_encrypt() => password\_hash()

Password\_check() => password\_verify()

* Code is already available as a libray
* Htpps://github.com/ircmaxell/password\_compat