What algorithm should I use to hash passwords into my database? [duplicate]

Asked 16 years, 3 months ago Modified 2 years ago Viewed 33k times

24

This question already has answers here:

Secure Password Hashing [closed] (9 answers)

Closed 8 years ago.

Is there *anything* available that isn't trivially breakable?

security encryption passwords hash

Share

Improve this question

Follow

asked Sep 22, 2008 at 18:41

Dustman

5,263 ● 10 ● 34 ● 40

11 Note for future readers: This information is dangerously out of date. Read this question on the IT Security Stack Exchange site for up-to-date info. – Brendan Long Sep 20, 2012 at 21:25

11 Answers

Sorted by: Highest score (default)

> *This 2008 answer is now dangerously out of date.* SHA (all variants) is now trivially breakable, and best practice is now (as of Jan 2013) to use a key-stretching hash (like PBKDF2) or ideally a RAM intensive one (like Bcrypt) and to add a per-user salt too.
>
> Points 2, 3 and 4 are still worth paying attention to.
>
> See the IT Security SE site for more.

Original 2008 answer:

1. Use a proven algorithm. SHA-256 uses 64 characters in the database, but with an index on the column that isn't a problem, and it is a proven hash and more reliable than MD5 and SHA-1. It's also implemented in most languages as part of the standard security suite. However don't feel bad if you use SHA-1.
2. Don't just hash the password, but put other information in it as well. You often use the hash of "username:password:salt" or similar, rather than just the password, but if you play with this then you make it even harder to run a dictionary attack.
3. Security is a tough field, do not think you can invent your own algorithms and protocols.

4. Don't write logs like "[AddUser] Hash of GeorgeBush:Rep4Lyfe:ASOIJNTY is xyz"

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Sep 22, 2008 at 18:51

JeeBee

17.5k ● 5 ● 52 ● 60

8 While likely accurate at the time of posting, it is now strongly recommended that you SHA-1 not be used. – Brian Apr 26, 2012 at 16:16

3 This answer was accurate at the time, but cloud processing power means that heavy duty brute force attacks are now cheap. SHA-1 and MD5 are now trivially breakable. These days I'd use SHA-256 minimum and always add a sizable salt. If you need future proofing I'd look into a deliberately slow and processor intensive hash like bcrypt or scrypt. – Keith Jul 19, 2012 at 10:16

2 Shocking how such a short period of time has led to believed-to-be-secure algorithms to be considered weak simply due to brute force attack improvements. – JeeBee Oct 15, 2013 at 9:10

Actually, this was already bad advice when this was written. Password hashing scheme had used work factors for decades! Check Unix crypt as an example. – Erwan Legrand Oct 30, 2017 at 12:34

First rule of cryptography and password storage is "*don't invent it yourself*," but if you must here is the absolute

minimum you must do to have any semblance of security:

Cardinal rules:

1. Never store a plain text password (which means you can never display or transmit it either.)
2. Never transmit the stored representation of a password over an unsecured line (either plain text, encoded or hashed).
3. *Speed is your enemy.*
4. Regularly reanalyze and *improve your process* as hardware and cryptanalysis improves.
5. Cryptography and process is a *very small* part of the solution.
6. Points of failure include: storage, client, transmission, processing, user, legal warrants, intrusion, and administrators.

Steps:

1. Enforce some reasonable minimum password requirements.
2. Change passwords frequently.
3. Use the strongest hash you can get - **SHA-256** was suggested here.
4. Combine the password with a *fixed salt* (same for your whole database).

5. Combine the result of previous step with a *unique salt* (maybe the username, record id, a guid, a long random number, etc.) that is stored and attached to this record.
6. Run the hash algorithm multiple times - *like 1000+ times*. Ideally include a different salt each time with the previous hash. Speed is your enemy and multiple iterations reduces the speed. Every so often double the iterations (this requires capturing a new hash - do it next time they change their password.)

Oh, and unless you are running SSL or some other line security then don't allow your password to be transmitted in plain text. And if you are only comparing the final hash from the client to your stored hash then don't allow that to be transmitted in plain text either. You need to send a nonce (number used once) to the client and have them hash that with their generated hash (using steps above) hash and then they send you that one. On the server side you run the same process and and see if the two one time hashes match. Then dispose of them. There is a better way, but that is the simplest one.

answered Sep 23, 2008 at 18:58

Jim McKeeth

38.7k ● 25 ● 124 ● 199

Why use a site wide salt and also a user unique salt? Wouldn't a single large random salt value be sufficient?
– Jason Fritcher Jun 17, 2009 at 3:26

2 A single salt (of any size) means that if they generate a rainbow table (hash dictionary) once, it is good for every user in the database, so they are much more likely to find a hit quicker. If each user has a different salt then it means a new lookup table needs to be generated for each user.
– Jim McKeeth Jun 17, 2009 at 3:39

I understand why its bad to use just a single salt for all users. I wanted to find out what the idea is behind using a single site wide salt plus a user unique salt when, at least to me, just a user unique salt appears to be sufficient. – Jason Fritcher Jun 17, 2009 at 14:48

The additional site wide salt is more important if you are using the username or some other weak salt as your user specific salt. For example, if a user has the same username and password on another site then you will have the same hash on both sites. Beyond that it is just an incremental improvement. – Jim McKeeth Jul 17, 2009 at 5:18

Random salts are best. Change them every time the user changes/creates a password. You can store the extra 3-4 characters in the same field as the password hash, separating them with, e.g. `:`. So your field goes from `aabbccddeeaabbccddeeaabbccddeeaabbccddee` to `sLt:fb1337ce1afb1337ce1afb1337ce1afb1337ce1a`. Swap the order of the two subfields if you want, but the point is, it's cheap to parse later, and increases the security dramatically. – maxwellb Aug 7, 2009 at 19:52

18

CodingHorror had a great article on this last year. The recommendation at the end of the article is bcrypt.

Also see: https://security.stackexchange.com/questions/4781/do-any-security-experts-recommend-bcrypt-for-password-storage/6415#6415

Share Improve this answer Follow

edited Nov 22, 2022 at 22:24

answered Sep 22, 2008 at 18:43

Lou Franco

89k ● 14 ● 136 ● 198

> Here is an implementation in .net: derekslager.com/blog/posts/2007/10/… – Yaakov Ellis Dec 18, 2008 at 8:04

8

The aforementioned algorithms are cryptographically secure hashing algorithms (but MD5 isn't considered to be secure today).

However there are algorithms, that specifically created to derive keys from passwords. These are the key derivation functions. They are designed for use with symmetric ciphers, but they are good for storing password too. PBKDF2 for example uses salt, large number of iterations, and a good hash function. If you have a library,

what implements it (e.g. .NET), I think you should consider it.

Share Improve this answer

Follow

answered Sep 22, 2008 at 19:13

KovBal

2,177 ● 2 ● 18 ● 32

I think there are good uses for PBKDF's, but if you are implementing a server application with password-protection, the password needs to be transformed before it goes over the wire. Often PBKDF's will be used to *generate a key* for use in, e.g. symmetric encryption. A password-protected forum usually needs to use cheap algorithms with pretty good collision-resistance, as the result of the transformation is essentially the "password" to check against the database. Increasing randomness and decreasing likelihood of collisions are how to increase security here. Random salt+good hash should work – maxwellb Aug 7, 2009 at 19:56

@mpbloch : You can't transform the password before it goes over the wire. You can use SSL to protect it. You need to hash the passwords because of the rainbow table attacks. And it's only possible if you hash it on the server. – KovBal Aug 11, 2009 at 13:26

6

Add a unique salt to the hashed password value (store the salt value in the db). When a unique salt is used the benefit of using a more secure algorithm than SHA1 or MD5 is not really necessary (at that point it's an incremental improvement, whereas using a salt is a monumental improvement).

Share Improve this answer

Follow

answered Sep 22, 2008 at 18:55

Wedge

19.8k ● 7 ● 49 ● 71

6

Use a strong crytographic hash function like MD5 or SHA1, but make sure you use a good salt, otherwise you'll be susceptible to rainbow table attacks.

Share Improve this answer

Follow

edited Sep 22, 2008 at 18:58

answered Sep 22, 2008 at 18:45

Adam Rosenfield

399k ● 101 ● 522 ● 597

5

Update Jan 2013

The original answer is from 2008, and things have moved a bit in the last 5 years. The ready availability of cloud computing and powerful parallel-processor graphics cards means that passwords up to 8 or 9 characters hashed as MD5 or SHA1 are now trivially breakable.

Now a long salt is a must, as is something tougher like SHA512.

However all SHA variant hashes are designed for communication encryption - messages back and forth where every message is encrypted, and for this reason they are designed to be *fast*.

In the password hashing world this design is a big disadvantage as the quicker the hash is the generate the less time it takes to generate large numbers of hashes.

A fast hash like SHA512 can be generated millions, even billions of times a second. Throw in cheap parallel processing and every possible permutation of a password becomes an absolute must.

Key-stretching is one way to combat this. A key-stretching algorithm (like PBKDF2) applies a quicker hash (like SHA512) thousands of times, typically causing the hash generation to take 1/5 of a second or so. Someone logging in won't notice, but if you can only generate 5 hashes per second brute force attacks are much tougher.

Secondly there should *always* be a per-user random salt. This can be randomly generated as the first *n* bytes of the hash (which are then stripped off and added to the password text to be checked before building the hashes to compare) or as an extra DB column.

So:

> What algorithm should I use to hash passwords into my database?

- *Key-stretching* to slow down hash generation. I'd probably go with PBKDF2.
- *Per-user salt* means a new attack per user, and some work figuring out how to get the salt.

Computing power and availability are going up exponentially - chances are these rules will change again in another 4 years. If you need future-proof security I'd investigate bcrypt/scrypt style hashes - these take the slower key-stretching algorithms and add a step that uses a lot of RAM to generate the hash. Using so much RAM reduces the effectiveness of cheap parallel processors.

Original Sept 2008 (left in so comments make sense)

MD5+salt or SHA1+salt is not 'trivially breakable' - most hacks depend on huge rainbow tables and these become less useful with a salt `[update, now they are]`.

MD5+salt is a relatively weak option, but it isn't going to be easily broken `[update, now it is very easy to break]`.

SHA2 goes all the way up to 512 - that's going to be pretty impossible to crack with readily available kit `[update, pretty easy up to 9 char passwords now]` - though I'm sure there's a Cray in some military bunker somewhere that can do it `[You can now rent this 'Cray' from Amazon]`

edited Jan 23, 2013 at 12:12

answered Sep 22, 2008 at 18:57

Keith
155k ● 82 ● 306 ● 446

SHA2 is surprisingly fast. Definitly much faster even than AES... Even hashing large amounts of data many times, I find it highly unlikely that performance will be degraded any significant amount. – AviD Sep 22, 2008 at 19:10

1

MD5 or SHA in combination with a randomly generated salt value for every entry

Share Improve this answer

Follow

answered Sep 22, 2008 at 18:45

Vasil

38k ● 27 ● 93 ● 116

1

as mentioned earlier simple hashing algorithms should not be used here is reason why :

http://arstechnica.com/security/2012/08/passwords-under-assault/

so use something else such as http://msdn.microsoft.com/en-us/library/system.security.cryptography.rfc2898derivebytes.aspx

Share Improve this answer

Follow

answered Jan 22, 2013 at 12:49

zebra

1,338 ● 1 ● 13 ● 27

0

All hashing algorithms are vulnerable to a "dictionary attack". This is simply where the attacker has a very large dictionary of possible passwords, and they hash all of them. They then see if any of those hashes match the hash of the password they want to decrypt. This technique can easily test millions of passwords. This is why you need to avoid any password that might be remotely predictable.

But, if you are willing to accept the threat of a dictionary attack, MD5 and SHA1 would each be more than adequate. SHA1 is more secure, but for most applications this really isn't a significant improvement.

Share Improve this answer

Follow

answered Sep 22, 2008 at 18:47

sanity

35.7k ● 43 ● 140 ● 228

If you use salt you will make dictionary attacks much more difficult. – Kip Sep 22, 2008 at 18:55

"much more difficult"? Proper salt makes a dictionary attack impossible because there's not enough storage on the planet. – erickson Sep 22, 2008 at 22:40

3 Salt isn't against dictionary attack, but to defeat rainbow tables. The two concept are very different. – KovBal Jul 17, 2009 at 15:54

+1 to KovBal. Also, using a k-bit hash function, a "dictionary attack" against all possible passwords is 2^k many "words". Using a non-alphanumeric (essentially) alphabet already

increases the search space exponentially. Compare "provide alphanumeric password up to 8 characters" with "160-bit integer". Which one has a larger dictionary? – maxwellb Aug 7, 2009 at 20:01

Salt is to defeat dictionary attacks. A rainbow table is a specific form of dictionary attack. It's not a different concept. – erickson Oct 27, 2009 at 16:51

MD5 / SHA1 hashes are both good choices. MD5 is slightly weaker than SHA1.

-3

Share Improve this answer

Follow

answered Sep 22, 2008 at 18:43

JeffFoster

400 ● 1 ● 3

MD5/SHA1 are already cracked don't use them now , use SHA2 with salt or better off use Bcrypt OR Scrypt OR PBKBF2 , remember a slow hashing algorithm provide some degree of safeguard against brute force attacks. – Roshan Jul 23, 2022 at 10:46