Unique key generation

Asked 16 years, 3 months ago Modified 1 year, 1 month ago Viewed 17k times

Part of PHP Collective

12

I looking for a way, specifically in PHP that I will be guaranteed to always get a unique key.

I have done the following:

```
strtolower(substr(crypt(time()), 0, 7));
```

But I have found that once in a while I end up with a duplicate key (rarely, but often enough).

I have also thought of doing:

```
strtolower(substr(crypt(uniqid(rand(), true)), 0, 7));
```

But according to the PHP website, uniqid() could, if uniqid() is called twice in the same microsecond, it could generate the same key. I'm thinking that the addition of rand() that it rarely would, but still possible.

After the lines mentioned above I am also remove characters such as L and O so it's less confusing for the user. This maybe part of the cause for the duplicates, but still necessary.

One option I have a thought of is creating a website that will generate the key, storing it in a database, ensuring it's completely unique.

Any other thoughts? Are there any websites out there that already do this that have some kind of API or just return the key. I found http://userident.com but I'm not sure if the keys will be completely unique.

This needs to run in the background without any user input.

PHP php web-services security passwords

Share

Improve this question

Follow

edited Oct 20, 2008 at 19:12

asked Sep 10, 2008 at 20:17

Darryl Hein

145k ● 96 ● 223 ● 263

Why do the passwords need to be completely unique? – Michael Haren Sep 10, 2008 at 20:24

I suppose crypt is only because you need to crypt value, it has got nothing to do with creating unique value, hasn't it?! – Marco Demaio Mar 18, 2010 at 19:22

See also: PHP function to generate v4 UUID. – Jack Nov 25, 2014 at 3:12

13 Answers

Sorted by: Highest score (default)

19

There are only 3 ways to generate unique values, rather they be passwords, user IDs, etc.:

1. Use an effective GUID generator - these are long and cannot be shrunk. If you only use part **you FAIL**.
2. At least part of the number is sequentially generated off of a single sequence. You can add fluff or encoding to make it look less sequential. Advantage is they start short - disadvantage is they require a single source. The work around for the single source limitation is to have numbered sources, so you include the [source #] + [seq #] and then each source can generate its own sequence.
3. Generate them via some other means and then check them against the single history of previously generated values.

Any other method is not guaranteed. Keep in mind, fundamentally you are generating a binary number (it is a computer), but then you can encode it in Hexadecimal, Decimal, Base64, or a word list. Pick an encoding that fits your usage. Usually for user entered data you want some variation of Base32 (which you hinted at).

Note about GUIDS: They gain their strength of uniqueness from their length and the method used to generate them. *Anything less than 128-bits is not secure.* Beyond random number generation there are characteristics that go into a GUID to make it more unique. Keep in mind they are only practically unique, not completely unique. It is possible, although practically impossible to have a duplicate.

Updated Note about GUIDS: Since writing this I learned that many GUID generators use a cryptographically secure random number generator (difficult or impossible to predict the next number generated, and a not likely to repeat). There are actually 5 different UUID algorithms. Algorithm 4 is what Microsoft currently uses for the Windows GUID generation API. A GUID is Microsoft's implementation of the UUID standard.

Update: If you want 7 to 16 characters then you need to use either method 2 or 3.

Bottom line: Frankly there is no such thing as completely unique. Even if you went with a sequential generator you would eventually run out of storage using all the atoms in the universe, thus looping back on yourself and repeating. Your only hope would be the heat death of the universe before reaching that point.

Even the best random number generator has a possibility of repeating equal to the total size of the random number you are generating. Take a quarter for example. It is a completely random bit generator, and its odds of repeating are 1 in 2.

So it all comes down to your threshold of uniqueness. You can have 100% uniqueness in 8 digits for 1,099,511,627,776 numbers by using a sequence and then base32 encoding it. Any other method that does not involve checking against a list of past numbers only has odds equal to n/1,099,511,627,776 (where n=number of previous numbers generated) of not being unique.

Share
Improve this answer
Follow

edited Jul 30, 2011 at 18:33

answered Sep 10, 2008 at 21:06

Jim McKeeth
38.7k ● 25 ● 124 ● 199

Hello, do you have any further reading about number 2, I'd like to use this kind of code generation using different "sources", Thanks. – Jon Sep 14, 2012 at 9:48

@Jon: I don't have any reading on this, but essentially you concatenate the a source identifier with a sequence from that source. So you know it is unique. For example, if you had two sources, then you would have a sequence of A1, A2, A3, A4 .. . A999 and another sequence of B1, B2, B3, B4 . .. B999. Since a ID from source A will never start with B, then you know there will never be a collision. – Jim McKeeth Sep 15, 2012 at 4:43

k thank you, could the source identifier be an id for a db table so that when a code is entered information could be gather from a record in a table? – Jon Sep 19, 2012 at 10:40

1

Any algorithm will result in duplicates.

Therefore, might I suggest that you use your existing algorithm* and simply check for duplicates?

*Slight addition: If `uniqid()` can be non-unique based on time, also include a global counter that you increment after every invocation. That way something is different even in the same microsecond.

Share Improve this answer Follow

answered Sep 10, 2008 at 20:30

Jason Cohen
83k ● 26 ● 110 ● 114

GUID (or UUIDS) and sequences are the only two ways to avoid duplicates, but you are correct for every other algorithm mentioned here. – Jim McKeeth Sep 10, 2008 at 21:36

If arbitrary identifier lengths are allowed, I agree. If he wants to keep to 8 chars as in his examples, it's just not enough. – Jason Cohen Sep 10, 2008 at 21:59

8 characters is enough for exactly 1,099,511,627,776 unique passwords if you use Base32, which is respectable for manually entered data (32^8) This makes no allowance for verification, nor does it exclude patterns like 00000000. – Jim McKeeth Sep 10, 2008 at 22:40

Collisions are due to his algorithm or his encoding (maybe only using Decimal, which is only 100,000,000 possibilities. Centralized sequential generation is the only solution that will work, or checking for duplicates like you suggested. – Jim McKeeth Sep 10, 2008 at 23:10

0

Without writing the code, my logic would be:

Generate a random string from whatever acceptable characters you like.
Then add half the date stamp (partial seconds and all) to the front and the other half to the end (or somewhere in the middle if you prefer).

Stay JOLLY!
H

Share Improve this answer Follow

answered Sep 10, 2008 at 20:20

Humpton
1,519 ● 5 ● 18 ● 26

1 This is close, but depending on your random routine only has just over microsecond accuracy. As he mentioned, uniqid failed for the same possibility. – Jim McKeeth Sep 10, 2008 at 21:28

0

If you use your original method, but add the username or emailaddress in front of the password, it will always be unique if each user only can have 1 password.

Share Improve this answer Follow

answered Sep 10, 2008 at 20:22

Espo
41.9k ● 21 ● 136 ● 161

0

I usually do it like this:

```
$this->password = '';
```

```
for($i=0; $i<10; $i++)
{
    if($i%2 == 0)
        $this->password .= chr(rand(65,90));
    if($i%3 == 0)
        $this->password .= chr(rand(97,122));
    if($i%4 == 0)
        $this->password .= chr(rand(48,57));
}
```

I suppose there are some theoretical holes but I've never had an issue with duplication. I usually use it for temporary passwords (like after a password reset) and it works well enough for that.

Share Improve this answer Follow

answered Sep 10, 2008 at 20:23

Mark Biek
151k ● 54 ● 158 ● 201

0

I'm still not seeing why the passwords have to be unique? What's the downside if 2 of your users have the same password?

This is assuming we're talking about passwords that are tied to userids, and not just unique identifiers. If *that's* what you're looking for, why not use GUIDs?

Share Improve this answer Follow

answered Sep 10, 2008 at 20:51

zigdon
15k ● 6 ● 37 ● 54

> because it may not always be a password, but instead their actual username/login id or say a transaction id. Both of these are basically a username so they have to be unique. I have the database setup to only accept unique passwords, but again, I can't assume I'm dealing with a database that I have access to. The other thing is I only want 7 to maybe 15 characters. GUIDs are typically much longer and therefore will end up not being unique quite easily.
> – Darryl Hein Sep 10, 2008 at 20:58

0

You might be interested in Steve Gibson's over-the-top-secure implementation of a password generator (no source, but he has a detailed description of how it works) at https://www.grc.com/passwords.htm.

The site creates huge 64-character passwords but, since they're completely random, you could easily take the first 8 (or however many) characters for a less secure but "as random as possible" password.

EDIT: from your later answers I see you need something more like a GUID than a password, so this probably isn't what you want...

Share

Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 • 1

answered Sep 10, 2008 at 20:27

dF.

75.7k • 31 • 135 • 137

0

I do believe that part of your issue is that you are trying to us a singular function for two separate uses... passwords and transaction_id

these really are two different problem areas and it really is not best to try to address them together.

Share Improve this answer Follow

answered Sep 11, 2008 at 21:11

Laith

349 • 1 • 7

0

I recently wanted a quick and simple random unique key so I did the following:

```
$ukey = dechex(time()) . crypt( time() . md5(microtime() + mt_rand(0, 100000))
);
```

So, basically, I get the unix time in seconds and add a random md5 string generated from time + random number. It's not the best, but for low frequency requests it is pretty good. It's fast and works.

I did a test where I'd generate thousands of keys and then look for repeats, and having about 800 keys per second there were no repetitions, so not bad. I guess it totally depends on mt_rand()

I use it for a survey tracker where we get a submission rate of about 1000 surveys per minute... so for now (crosses fingers) there are no duplicates. Of course, the rate is not constant (we get the submissions at certain times of the day) so this is not fail proof nor the best solution... the tip is using an incremental value as part of the key (in my case, I used time(), but could be better).

Share Improve this answer Follow

answered Sep 15, 2009 at 11:58

Vince

929 • 8 • 16

1 I don't undertstand why you need to md5 values. MD5 is a one way hashing that creates a signature. Be careful that different input values might result in creating the same signature, MD5 functions are triggered to attempt to return different results when input values are slightly different not completly different. So by adding MD5 I think you increase the probability to get duplicated values. – Marco Demaio Mar 18, 2010 at 19:27

Ingoring the crypting part that does not have much to do with creating a unique value I usually use this one:

```
function GetUniqueValue()
{
   static $counter = 0; //initalized only 1st time function is called
   return strtr(microtime(), array('.' => '', ' ' => '')) . $counter++;
}
```

When called in same process $counter is increased so value is always unique in same process.

When called in different processes you must be really unlucky to get 2 microtime() call with the same values, think that microtime() calls usually have different values also when called in same script.

Share Improve this answer Follow

answered Mar 18, 2010 at 19:49

Marco Demaio
34.3k • 33 • 130 • 161

You may be interested in this article which deals with the same issue: [GUIDs are globally unique, but substrings of GUIDs aren't](#).

> The goal of this algorithm is to use the combination of time and location ("space-time coordinates" for the relativity geeks out there) as the uniqueness key. However, timekeeping is not perfect, so there's a possibility that, for example, two GUIDs are generated in rapid succession from the same machine, so close to each other in time that the timestamp would be the same. That's where the uniquifier comes in.

Share

Improve this answer

Follow

edited Nov 11, 2023 at 10:03

dumbass
27.2k • 4 • 35 • 72

answered Sep 10, 2008 at 20:22

Frank Krueger
70.9k • 48 • 164 • 211

-1

I usually do a random substring (randomize how many chars between 8 an 32, or less for user convenience) or the MD5 of some value I have gotten in, or the time, or some combination. For more randomness I do MD5 of come value (say last name) concatenate that with the time, MD5 it again, then take the random substring. Yes, you *could* get equal passwords, but its not very likely at all.

Share Improve this answer Follow

answered Sep 10, 2008 at 20:26

Adam Lerman

3,399 ● 9 ● 43 ● 53

> If you take a substring of a cryptographically secure hash then you defeat the uniqueness of the hash. No portion of a hash is more secure or random than any other. It is just as likely to get equal passwords as using a regular call to a random number generator. – Jim McKeeth Sep 10, 2008 at 21:35

-1

As Frank Kreuger commented, go with a GUID generator.

Like this one

Share Improve this answer Follow

answered Sep 10, 2008 at 20:31

George Mauer

122k ● 139 ● 395 ● 626