

Forum:[Hack Forums](#)

Thread: [Whats the best way to generate a random number?](#)

CodeBinary - April 11, 2016, 8:41 pm

For some reason it generates over 50 a lot more often then under. Is there a way to make it evenly happen or to go under more often then over.

I am using php7's random_int is that secure? I was told to seed it what is that and how do I do it?

Team4o4 - April 11, 2016, 10:11 pm

(Apr 11, 2016 08:41 PM)Dashy Wrote: [\[CITE\]](#) For some reason it generates over 50 a lot more often then under. Is there a way to make it evenly happen or to go under more often then over.

I am using php7's random_int is that secure? I was told to seed it what is that and how do I do it?
PHP's built in rand function works quite well, and after personally running a test 1000 times, I found it to be quite accurate for random-ness. I simply charted how many times each number in my range came out, and yes some had less while others had more - but for the most part it was evenly distributed. Therefore, I like to use:

PHP Code
<?php
rand(\$min, \$max);

Pillows - April 11, 2016, 10:17 pm

I've been hearing that mt_rand() is better:

http://www.w3schools.com/php/func_math_mt_rand.asp

carbos - April 12, 2016, 7:59 am

(Apr 11, 2016 10:17 PM)Pillows Wrote: [\[CITE\]](#) I've been hearing that mt_rand() is better:

http://www.w3schools.com/php/func_math_mt_rand.asp
* mt_rand() is better

Sakuto - April 12, 2016, 8:25 am

rand() generate a [PRNG](#) as every other random function. You could ask the user to move his mouse and generate something related to his mouse movement if you really want something random :)

ASDA - April 12, 2016, 8:45 am

(Apr 11, 2016 10:11 PM)Team4o4 Wrote: [\[CITE\]](#) PHP's built in rand function works quite well, and after personally running a test 1000 times
that's literally the equivalent of saying you drove a car 5 feet and you found it rode smoothly and that the car runs quite well.

OP: read up on Mersenne Twister and lib rand().

tl;dr neither. if you need SECURE random numbers you should use a true source of randomness (atmospheric noise, an entropy key, mouse input)

/dev/random could work too in a lot of cases where you need true randomness.

PHP Code

```
<?php
hexdec(bin2hex(file_get_contents('/dev/random', false, null, -1, 4)));
```

mothh - April 12, 2016, 9:47 am

(Apr 12, 2016 07:59 AM)carbos Wrote: [\[CITE\]](#) * mt_rand() is better

It isn't, apparently it tends to go with more even number than uneven numbers or the other way around.

ASDA - April 14, 2016, 3:25 am

(Apr 12, 2016 09:47 AM)mothh Wrote: [\[CITE\]](#) It isn't, apparently it tends to go with more even number than uneven numbers or the other way around.

don't know where you've read that shit. mersenne twister is infinitely times better than the libc random.

https://en.wikipedia.org/wiki/Mersenne_Twister

Sakuto - April 14, 2016, 9:02 am

Probability of getting even or odd number is basically 50%... So yeah, maybe you could get even more often than odd, this is just probability. Do it on a large scale (x > 1000000) and check the result.

mothh - April 14, 2016, 1:56 pm

(Apr 14, 2016 03:25 AM)ASDA Wrote: [\[CITE\]](#) don't know where you've read that shit. mersenne twister is infinitely times better than the libc random.

https://en.wikipedia.org/wiki/Mersenne_Twister

I tested it and it seems that the `rand()` function divides it more evenly than the `mt\_rand()` function.

Sakuto - April 14, 2016, 1:58 pm

(Apr 14, 2016 01:56 PM)mothh Wrote: [\[CITE\]](#) I tested it and it seems that the `rand()` function divides it more evenly than the `mt\_rand()` function.

How did you tested it? On which scale?

RumbleFrog - April 14, 2016, 6:50 pm

If you're on PHP7, I'd recommend random_int, I believe it's more random than other functions.

Think of the function as a dice. But with other function, one side is always more weighted, which is less random, random_int is the most balanced IMO.

Br4v0 - April 15, 2016, 1:10 am

normally i use this function

PHP Code

```
<?php
function RandomString($length = 10) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $charactersLength = strlen($characters);
    $randomString = "";
    for ($i = 0; $i < $length; $i++) {
```

```

    $randomString .= $characters[rand(0, $charactersLength - 1)];
}
return $randomString;
}

```

_____ - April 15, 2016, 1:11 am

PHP does not generate something completely random. You should try something like random.org.

Remix - April 15, 2016, 1:54 am

(Apr 15, 2016 01:10 AM)Br4v0 Wrote: [\[CITE\]](#) normally i use this function

PHP Code

```

<?php
function RandomString($length = 10) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $charactersLength = strlen($characters);
    $randomString = "";
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $characters[rand(0, $charactersLength - 1)];
    }
    return $randomString;
}

```

1. Not what he asked for
2. <http://php.net/manual/en/function.openssl-random-pseudo-bytes.php>

OP use the function [mt_rand](#)

It produces a better random value than rand and it's 4 times faster

Punkbastard - April 15, 2016, 2:00 am

For true randomness use /dev/urandom, the preferred source of cryptographic randomness on UNIX-like systems.

I personally use the php function [mcrypt_create_iv\(\\$length, MCRYPT_DEV_URANDOM\)](#) for randomness.

ASDA - April 15, 2016, 3:21 am

(Apr 15, 2016 02:00 AM)Punkbastard Wrote: [\[CITE\]](#) For true randomness use /dev/urandom, the preferred source of cryptographic randomness on UNIX-like systems.

I personally use the php function [mcrypt_create_iv\(\\$length, MCRYPT_DEV_URANDOM\)](#) for randomness.

you mean /dev/random.

(Apr 14, 2016 01:56 PM)mothh Wrote: [\[CITE\]](#) I tested it and it seems that the `rand()` function divides it more evenly than the `mt\_rand()` function.

trust me when i say this: your sample size was fucking nothing.

Sakuto - April 15, 2016, 3:24 am

He tested it on 10.000 test! It's enough :O

mothh - April 15, 2016, 3:48 am

(Apr 15, 2016 03:21 AM)ASDA Wrote: [\[CITE\]](#) you mean /dev/random.

trust me when i say this: your sample size was fucking nothing.
It was just an observation, scaling it up yielded the same results pretty much.

Sakuto - April 15, 2016, 3:56 am

Tested with 10.000.000 try, and the proportion of result was pretty much equivalent.

ASDA - April 15, 2016, 4:00 am

(Apr 15, 2016 03:48 AM)mothh Wrote: [\[CITE\]](#) It was just an observation, scaling it up yielded the same results pretty much.

Whatever clib you're using must be broken then. Only explanation. I very very highly doubt a proper implementation of libc random will produce numbers that have a pattern visible to the human eye.

Would love to see your generated data and derived graphs.

Punkbastard - April 15, 2016, 4:17 am

(Apr 15, 2016 03:21 AM)ASDA Wrote: [\[CITE\]](#) you mean /dev/random.

No, I mean /dev/urandom.

A very usefull article: [Secure random numbers for PHP developers](#)

ASDA - April 15, 2016, 4:51 am

(Apr 15, 2016 04:17 AM)Punkbastard Wrote: [\[CITE\]](#) No, I mean /dev/urandom.

A very usefull article: [Secure random numbers for PHP developers](#)

no you fucking dont. you mean /dev/random. don't tell me i'm wrong. i'm not even clicking that link.

Punkbastard - April 15, 2016, 5:11 am

(Apr 15, 2016 04:51 AM)ASDA Wrote: [\[CITE\]](#) no you fucking dont. you mean /dev/random. don't tell me i'm wrong. i'm not even clicking that link.

I mean /dev/urandom/, not /dev/random, how can you know what I mean?

There's a difference between the two, but since you're not willing to read that article, I'll quote it for you:

Quote:/dev/random vs. /dev/urandom?

Unfortunately Linux comes with two interfaces for random numbers. I say unfortunately, because the choice between them feeds confusion. According to the experts and making long story short, both of them are secure (in any practical purpose), but the blocking nature of /dev/random makes it usually unusable in interactive web application usage scenario.

Or, to say it in less words: use /dev/urandom and be happy; use /dev/random and be sorry.

I was just giving my advise by telling the OP to use /dev/urandom.

I was not telling /dev/random was more insecure, but just because of it's blocking nature I prefer /dev/urandom.

ASDA - April 15, 2016, 5:52 am

(Apr 15, 2016 05:11 AM)Punkbastard Wrote: [\[CITE\]](#) I mean /dev/urandom/, not /dev/random, how can you know what I mean?

There's a difference between the two, but since you're not willing to read that article, I'll quote it for you:

I was just giving my advise by telling the OP to use /dev/urandom.

I was not telling /dev/random was more insecure, but just because of it's blocking nature I prefer /dev/urandom.

not that your site gets enough traffic to drain /dev/random's buffer sorry not sorry. you cannot use the words "true randomness" and /dev/urandom in the same sentence. there's no two ways about it.

if you're draining /dev/random's buffer you need an entropy key. don't use /dev/urandom for "true"

random numbers.

sarciszewski - April 15, 2016, 12:11 pm

Are you all high? Just use `random_int()`, and if you're on PHP 5, use [random_compat](#) to polyfill it.
`rand()`, not even once.

(Apr 15, 2016 05:52 AM)ASDA Wrote: [CITE](#) don't use `/dev/urandom` for "true" random numbers. As the resident cryptography engineer, this is bullshit.

Quote:The Linux `/dev/urandom` manual page claims that without new entropy the user is "theoretically vulnerable to a cryptographic attack", but (as I've mentioned in various venues) this is a ludicrous argument—how can anyone simultaneously believe that we can't figure out how to deterministically expand one 256-bit secret into an endless stream of unpredictable keys (this is what we need from `urandom`), but

we can figure out how to use a single key to safely encrypt many messages (this is what we need from SSL, PGP, etc.)?

There are also people asserting that it's important for RNGs to provide "prediction resistance" against attackers who, once upon a time, saw the entire RNG state. But if the attacker sees the RNG state that was used to generate your long-term SSL keys, long-term PGP keys, etc., then what exactly are we gaining by coming up with unpredictable random numbers in the future?

Read [this](#) or STFU.

Haxalot - April 15, 2016, 12:31 pm

So much misinformation in one thread...

If you're using PHP 7, then the [random_int\(\)](#) function should be used when wanting to generate a cryptographically secure pseudorandom number (it's a CSPRNG). It pulls bytes from `/dev/urandom`, so it is considered secure. If it cannot find an appropriate source of randomness (such as the randomness source being exhausted), then [an exception will be thrown](#). The same applies to [random_bytes](#).

If you're on PHP 5.x, then use the [random_compat](#) library.

[mcrypt_create_iv](#) is fine for cryptographically secure randomness.

Definitely don't use [rand\(\)](#) or [mt_rand\(\)](#) if you want it to be cryptographically secure.

Avoid using [openssl_random_pseudo_bytes\(\)](#) if you care about security, because it internally uses a function that should not be used for cryptographic purposes.

And if you rolled your own solution (and are using it in a context that requires cryptographic security) then you're an idiot!

Also, fun fact of the day: due to a spelling mistake in PHP's source code, the Mersenne Twister algorithm (underlying `mt_rand()`) was mis-implemented. The randomness quality doesn't appear to be affected though...

EDIT: you beat me to it Scott!

ASDA - April 16, 2016, 9:52 am

(Apr 15, 2016 12:11 PM)sarciszewski Wrote: [\[CITE\]](#) Are you all high? Just use random_int(), and if you're on PHP 5, use [random_compat](#) to polyfill it.

rand(), not even once.

As the resident cryptography engineer, this is bullshit.

Read [this](#) or STFU.

well fuck me sideways, some random on hf just proved me, the openssl guys, the openbsd guys, the gpg guys (and all the gnu project guys) wrong. sorry about that! i'll go and file a bug report with openssl and gpg and tell them to go and switch to /dev/urandom because the HF resident cryptography engineer said it's safer.

cheers!

Haxalot - April 16, 2016, 12:06 pm

(Apr 16, 2016 09:52 AM)ASDA Wrote: [\[CITE\]](#) well fuck me sideways, some random on hf just proved me, the openssl guys, the openbsd guys, the gpg guys (and all the gnu project guys) wrong. sorry about that! i'll go and file a bug report with openssl and gpg and tell them to go and switch to /dev/urandom because the HF resident cryptography engineer said it's safer.

cheers!

You're using the straw man argument here. sarciszewski said (quite rightly) that your claim of /dev/urandom not being suitable for "true" (by which I assume you mean cryptographically secure) randomness is BS. The link given also does not state that it's safer over /dev/random either - it in fact states that both sources use the same CSPRNG, and so both have the same randomness.

Also, being a member of HF does not automatically make someone unqualified. There are actually a few quite intelligent members on here (though many don't seem particularly active because of the juvenility of the community).

sarciszewski - April 16, 2016, 2:38 pm

(Apr 16, 2016 12:06 PM)Haxalot Wrote: [\[CITE\]](#) You're using the straw man argument here. sarciszewski said (quite rightly) that your claim of /dev/urandom not being suitable for "true" (by which I assume you mean cryptographically secure) randomness is BS. The link given also does not state that it's safer over /dev/random either - it in fact states that both sources use the same CSPRNG, and so both have the same randomness.

Also, being a member of HF does not automatically make someone unqualified. There are actually a few quite intelligent members on here (though many don't seem particularly active because of the juvenility of the community).

I'm also SwiftOnSecurity's personal cryptographer ;)

<https://twitter.com/swiftonsecurity/status/709951114087510016>

Haxalot - April 16, 2016, 2:59 pm

(Apr 16, 2016 02:38 PM)sarciszewski Wrote: [\[CITE\]](#) I'm also SwiftOnSecurity's personal cryptographer ;)

<https://twitter.com/swiftonsecurity/status/709951114087510016>

Ha, that's pretty cool. Now hurry up and make Libsodium a core extension in PHP ;) Hopefully then we'll see it gain more exposure and usage amongst PHP developers...

ASDA - April 16, 2016, 3:04 pm

(Apr 16, 2016 02:59 PM)Haxalot Wrote: [\[CITE\]](#) Ha, that's pretty cool. Now hurry up and make Libsodium a core extension in PHP ;) Hopefully then we'll see it gain more exposure and usage amongst PHP developers...

What's wrong with <https://github.com/jedisct1/libsodium-php> ?

Haxalot - April 16, 2016, 3:18 pm

(Apr 16, 2016 03:04 PM)ASDA Wrote: [\[CITE\]](#) What's wrong with <https://github.com/jedisct1/libsodium-php> ?

Nothing - that is the PHP extension for lib sodium (currently in PECL). But by being apart of the core, extensions generally become more available, and given the not-so-good-looking state of cryptography in PHP at the moment, this change would be most welcome.

sarciszewski - April 16, 2016, 3:27 pm

(Apr 16, 2016 02:59 PM)Haxalot Wrote: [\[CITE\]](#) Ha, that's pretty cool. Now hurry up and make Libsodium a core extension in PHP ;) Hopefully then we'll see it gain more exposure and usage amongst PHP developers...

Sure thing. I still need to get Airship 0.2.0 out this weekend, which includes a decentralized public key audit log feature (Merkle trees ftw), as a prelude to the availability of the plugin system. I'll try to get the RFC draft written before Monday so it can be discussed. :)