

KNOWLEDGE BASE · PUBLIC DOCUMENT

Sending one-time codes (OTP) by SMS safely: expiry, attempt limits and logs

PHP & Development

CODE

RLX-AO-KB-2160

VERSION

v2026.10.16

COUNTRY · LANGUAGE

Angola · English

Interweb Angola

RLX Angola, Lda. · Group RLX Holdings

Tax ID (NIF) 500 260 5724 · Commercial registration no. 26464-25/250724

<https://www.meu.interweb.ao/knowledgebase/2160>



The short answer (for developers): a code sent by SMS is only as safe as the code around it. The rules: generate it with a secure random generator, store only a **hash** of it, with a **short validity**, an **attempt limit** and **single use**; limit how many SMS each number and each IP can trigger; and **never** write the code into logs, URLs or analytics tools. Remember too that the SMS itself can be read on a locked screen or diverted through a SIM swap: for accounts that matter, also offer an authenticator app (**authenticator app, SMS or security key**).

The risks and the defences

Risk	Defence
Guessing the code by brute force	An attempt limit per code; after it, the code stops working. Compare with <code>hash_equals</code> .
A code that still works hours later	Short validity, single use, and a new code cancels the previous one.
Your form used to fire off SMS in bulk (inflated traffic to numbers that pay whoever runs the scam)	Limits per number, per IP and per session; a captcha before sending; only the countries you serve; an alarm when volume is out of the ordinary.
The code shows up in logs or monitoring	Log “code sent to the number ending in...”, never the code. Check the HTTP client’s debug mode and any tool that stores request bodies.
A code requested for one thing used for another	Tie the code to the user, the purpose (log in, change password, confirm a payment) and the session.
A predictable code	<code>random_int</code> , which uses the system’s secure generator. Never <code>rand()</code> , <code>mt_rand()</code> or anything derived from the time.

How to do it, step by step (PHP)

1

Generate the code and store only the hash, with the secret read from the environment (**environment variables and secrets**):

```
$code = str_pad((string) random_int(0, 999999), 6, "0", STR_PAD_LEFT);  
$hash = hash_hmac("sha256", $code, getenv("OTP_SECRET"));  
// store: user, purpose, $hash, expires_at, attempts = 0
```

2

Before sending, check the limits: how many codes this number and this IP asked for recently. Above your limit, refuse with a neutral message and do not call the provider. Enforce a pause before “resend”. You can also hold them back at the web server: **rate limiting in nginx**.

3

Write the message: who is sending (your site’s name), the code, and a warning never to share it and that nobody from your team will ask for it. No accents that force Unicode, so it fits in a single part: **GSM-7 or Unicode**. A last line with @ followed by your site’s domain and # followed by the code lets compatible browsers fill it in by themselves.

4

When verifying: fetch that user’s latest unused code for that purpose; if it expired or ran out of attempts, refuse; add one attempt **before** comparing; compare in constant time and, if it matches, mark it as used:

```
$ok = hash_equals($row["hash"],  
hash_hmac("sha256", $attempt, getenv("OTP_SECRET")));
```

5

Close the leaks: send the request to the provider by POST (GET parameters end up in access logs), do not log that request’s body, and mask the number in your own logs.

6

Always answer the same way on the “send code” form, whether or not an account with that number exists, so you do not reveal who is a customer.

Never return the code to the browser “to make testing easier”, and never leave a fixed test code active in production. Both are classic mistakes, and either one cancels the whole check.

Many SMS providers have their own verification API, which generates, sends and checks the code and handles part of the limits. Using it moves part of this work to them; read what they keep and for how long. And do not wait for the delivery report before letting the user carry on: **delivery reports (DLR)**.

Want to limit requests to the code form at server level, on your VPS? Open a ticket and tell us the form’s address.

[Open a support ticket](#)

SEE ALSO

Authenticator app, SMS or security key: which two-step verification to use

Rate limiting in nginx: slowing down bots and password guessing

Environment variables and secrets for your application

Keeping passwords out of your PHP code: .env files and the right permissions

GSM-7 or Unicode in SMS: why one accent splits a message into more parts

SMS delivery reports (DLR): how your application knows whether a message arrived



Online version of this document

Interweb is a leading internet services company, with a mission to help businesses to be born and prosper through technology.

2020

FOUNDED

37,000+

CLIENTS

130+

SIGN-UPS A DAY

Interweb Angola

RLX Angola, Lda. • Group RLX Holdings

TAX ID (NIF)

500 260 5724

COMMERCIAL
REGISTRATION NO.

26464-25/250724

REGISTERED OFFICE

Rua Direita do Patriota, Bairro Dangereux
Belas, Luanda, Angola

CONTACT

info@interweb.ao • +244 942 011 191 • www.interweb.ao

OPENING HOURS

Monday to Friday - 08 AM to 04 PM
Saturday & Sunday - Closed