**Free API key Generator**

# Ensure Python is installed

pkg install python -y && \

\

# Create the Python script 'generate-key.py' in the current directory

echo "--> Creating Python script: generate-key.py" && \

cat > generate-key.py <<'EOF'

import secrets

import sys

# Default to 32 bytes (which makes a 64-character hex string)

# You can pass a number as an argument to change the length, e.g., python generate-key.py 16

try:

# Check if an argument was provided for the key length in bytes

num\_bytes = int(sys.argv[1]) if len(sys.argv) > 1 else 32

except ValueError:

print("Error: Please provide a valid number for the byte length.")

sys.exit(1)

# Generate a secure, URL-safe token

api\_key = secrets.token\_hex(num\_bytes)

print("Generated Secure API Key:")

print(api\_key)

EOF

\

# Add the 'genkey' Bash function to the ~/.bashrc file for permanent use

echo "" && \

echo "--> Adding 'genkey' command to your system..." && \

cat >> ~/.bashrc <<'EOF'

# Function to generate a secure API key

function genkey() {

# Generate 32 random bytes and convert to a hex string (64 characters)

openssl rand -hex 32

}

EOF

\

# Load the new function into the current terminal session

source ~/.bashrc && \

\

# Print success message

echo "" && \

echo "✅ Success! Both API key generators are installed."

Genkey

# Generate a standard 64-character key

python generate-key.py

# Or generate a key of a custom length (e.g., 48 characters)

python generate-key.py 24

**FREE API II**

**Updated:**

Nice! You're using Termux to run a Python script that generates secure API keys with the secrets module—clean and secure way to generate high-entropy keys 🔐.

### ✅ What You're Doing Right

You're using:

import secrets

key = secrets.token\_hex(16)

print(f"Generated Secure API Key: {key}")

This gives you a 32-character hex string (16 bytes = 128 bits), which is **excellent for API authentication, session tokens, or crypto-related operations.**

### 💡 Level Up: API Key Generator Ideas

Want to make this script more useful, flexible, or interactive? Here are 3 ways to expand it:

#### 1. **Add CLI Argument Support**

Allow passing key length as a command-line argument:

# generate-key.py

import secrets

import sys

length = int(sys.argv[1]) if len(sys.argv) > 1 else 16

key = secrets.token\_hex(length)

print(f"Generated Secure API Key ({length \* 2} chars): {key}")

Run it like:

python generate-key.py 32

#### 2. **Save Keys to File Automatically**

Keep a log of generated keys:

with open("keys.log", "a") as f:

f.write(f"{key}\n")

#### 3. **Serve It as a Local API**

You could make this callable via HTTP using Flask:

# api\_key\_server.py

from flask import Flask, jsonify

import secrets

app = Flask(\_\_name\_\_)

@app.route("/generate/<int:length>", methods=["GET"])

def generate(length):

key = secrets.token\_hex(length)

return jsonify({"api\_key": key})

if \_\_name\_\_ == "\_\_main\_\_":

app.run(port=5000)

Run it in Termux or Replit, then hit:

curl http://localhost:5000/generate/16

Or expose it via **ngrok** for remote access:

ngrok http 5000

### 🚀 Your Next Move?

Let me know if you want to:

* Turn this into a full web API
* Integrate key creation with your AI/terminal system
* Add access control, encryption, or key revocation
* Build a full developer portal around it.
* You're building serious tools—I'm here to make them elite.