



Keysmith

Authentication Token Generator

David Tucker

Sentinel

- tiered account manager
 - Groups of related keys are assigned a tier.
 - e.g. Bank Accounts: tier 2
 - Tiers specify key generation algorithm and expirations.
 - e.g. tier 2: concatenate 3 random words (good for 90 days)
 - Secure keys are generated automatically.
 - Keysmith

XKCD

| | | |
|--|--|--|
| <p> UNCOMMON (NON-GIBBERISH) BASE WORD ORDER UNKNOWN Tr0ub4dor &3 CAPS? COMMON SUBSTITUTIONS NUMERAL PUNCTUATION (YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.) </p> | <p> ~28 BITS OF ENTROPY $2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$ (PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.) DIFFICULTY TO GUESS: EASY </p> | <p> WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO? AND THERE WAS SOME SYMBOL... DIFFICULTY TO REMEMBER: HARD </p> |
| <p> correct horse battery staple FOUR RANDOM COMMON WORDS </p> | <p> ~44 BITS OF ENTROPY $2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$ DIFFICULTY TO GUESS: HARD </p> | <p> THAT'S A BATTERY STAPLE. CORRECT! DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT </p> |

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Why is this way best?

Say Bob uses a 10,000 word dictionary and a true random number generator to derive a passphrase of 4 concatenated words. Let's also assume Mallory (who knows how Bob's key was created) wants to break this key and has the ability to guess 2,000 keys per second.

How long (at most) will it take Mallory to brute force Bob's passphrase?

Odds of a correct guess:

$$\frac{1}{10^4} \cdot \frac{1}{10^4} \cdot \frac{1}{10^4} \cdot \frac{1}{10^4} = \frac{1}{10^{16}}$$

Time to get the correct key:



$$\left(\frac{10^{16} \text{ guesses}}{1}\right) \left(\frac{1 \text{ second}}{2,000 \text{ guesses}}\right) = 5(10^{12}) \text{ seconds} \\ \approx 158440 \text{ years}$$

Bottom line:

Most dictionaries have far more than 10,000 words, and most attackers will not know your word list or even your key generation algorithm!

Random Numbers

- atmospheric noise
- HTTP API
 - plain-text responses
- ranges
- example query
 - <http://www.random.org/integers/?num=3&min=0&max=43238&col=1&base=10&format=plain&rnd=new>

The screenshot shows the RANDOM.ORG website interface. At the top is a navigation bar with links: Home, Games, Numbers, Lists & More, Drawings, Web Tools, Statistics, Testimonials, Learn More, and Login. Below this is the site logo "RANDOM.ORG" and a search bar. A tagline "True Random Number Service" is visible. The main heading is "Random Integer Generator". A paragraph explains that the form allows generating random integers from atmospheric noise. The interface is divided into five parts: Part 1: The Integers (with input fields for number of integers, range, and format), Part 2: Choose Numeral System (with radio buttons for Hexadecimal, Decimal, Octal, and Binary), Part 3: Choose Output Format (with radio buttons for web page or text document), Part 4: Choose Randomization (with radio buttons for new, pregenerated by date, or pregenerated by ID), and Part 5: Go! (with a "Be patient!" message and buttons for "Get Numbers", "Reset Form", and "Switch to Simple Mode"). A note at the bottom states that numbers are picked independently and may contain duplicates, and mentions a "Sequence Generator" for randomized sequences.

Home Games Numbers Lists & More Drawings Web Tools Statistics Testimonials Learn More Login

RANDOM.ORG Search RANDOM.ORG Google Custom Search Search

True Random Number Service

Random Integer Generator

This form allows you to generate random integers. The randomness comes from atmospheric noise, which for many purposes is better than the pseudo-random number algorithms typically used in computer programs.

Part 1: The Integers

Generate random integers (maximum 10,000).

Each integer should have a value between and (both inclusive; limits $\pm 1,000,000,000$).

Format in column(s).

Part 2: Choose Numeral System

How should the integers be displayed?

☐ Hexadecimal (base 16)
☒ Decimal (base 10)
☐ Octal (base 8)
☐ Binary (base 2)

Part 3: Choose Output Format

How do you want the integers to be shown?

☒ On a nicely formatted web page (type text/html)
☐ As a bare-bones text document (type text/plain)

Part 4: Choose Randomization

Do you want a new randomization or one that was prepared earlier? [[explain this](#)]

☒ Generate your own personal randomization right now
☐ Use pregenerated randomization from
☐ Use pregenerated randomization based on persistent identifier (max 64 alphanumeric characters)

Part 5: Go!

Be patient! It may take a little while to generate your numbers...

Note: The numbers generated with this form will be picked independently of each other (like rolls of a die) and may therefore contain duplicates. There is also the [Sequence Generator](#), which generates randomized sequences (like raffle tickets drawn from a hat) and where

Implementation

- Imperative

- C



- Object-oriented

- Java



- Functional

- Haskell



- Logic

- Prolog

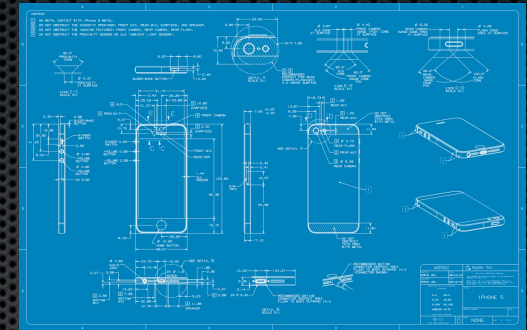


C



- error-checking, and a lot of it
- native command-line parser
 - `int getopt (int argc , char * const argv[] , const char * optstring);`
- native HTTP support
 - `cURL`
- no simple string manipulation
- generally unforgiving

Haskell



- what memory?
 - implementation specific (due to emphasized abstraction)
- native command-line parser
 - `System.Console.GetOpt`
- Hackage & Cabal
 - `Network.HTTP`, `RandomDotOrg`, etc.
- generally very forgiving
- excellent documentation
 - Hoogle, #haskell, more

Java



- container-based
- requires more meta-knowledge
 - e.g. `public void java.lang.System.out.println()`
- no native command-line parser (more graphical)
 - JCommander
- native support for CS RNG, kinda
 - `java.security.SecureRandom`
- heavyweight
 - setters & getters

Questions

