

PASSWORDTECH

OPEN-SOURCE PASSWORD GENERATOR & MANAGER

USER MANUAL

Version 3.3.0

Licensing Information
By using, copying, distributing or modifying “Password Tech” (“PwTech”; formerly known as “PWGen for Windows”) or a portion thereof you accept all terms and conditions contained in the file *license.txt* included in the package of this program.

Copyright Information
This software as a whole:
Copyright © 2002-2020 by Christian Thöing <c.thoeing@web.de>
Portions of this software:
Copyright © 2006-2014 by Brainspark B.V.: Implementations of AES, SHA1, and SHA-256 algorithms, base64 encoding from the “PolarSSL” library
Copyright © 1996-2015 by Markus F.X.J. Oberhumer: “minilzo” compression library
Copyright © 2000 by Arnold G. Reinhold: “diceware8k” word list
Copyright © 2005-2010 by Aha-Soft: Toolbar icons in main window (except “locked database” icon)
Copyright © 2008 by DryIcons: “Locked database” toolbar icon in main window
Copyright © 1994-2020 by Lua.org, PUC-Rio: Lua scripting language

World Wide Web
Homepage: http://pwgen-win.sourceforge.net/
Sourceforge.net project: http://sourceforge.net/projects/pwgen-win/

Contents

Introduction **4**
Program Features **5**
Unicode Support **6**
Supported Encodings 6
Password Generation with Password Tech – An Overview **8**
Step-by-Step Tutorial **10**
Profile Selection 10
Include Characters 10
Include Words 12
Format Password 14
Run Script 19
Advanced Password Options 19
Generate Multiple Passwords 23
Generate Single Passwords 24
Random Pool 25
Main Menu **26**
File 26
Profile 26
Profile | Profile Editor (<F10>) 27
Exit (<Alt>+<Q>) 27
Tools 28
Clear Clipboard (<F2>) 28
Encrypt/Decrypt Clipboard (<F3>/<F4>) 28
Create Random Data File (<F5>) 28
Create Trigram File (<F6>) 29
MP Password Generator (<F8>) 30
Deterministic Random Generator 35
Provide Additional Entropy 35
As Text (<F9>) 35
From File (<Ctrl>+<F9>) 36
Options 36
Configuration (<F11>) 36
General 36
Security 38
Hot Keys 39
Files 40
Updates 40
Language 41
Database 41
Always on Top 42
Save Settings on Exit 42
Hide Entropy Progress 42
Save Settings Now 42
Help 42
Open Manual (<F1>) 42
Visit Website 42
Get Translations 42

Donate....43
Enter Donor Key....43
Check for Updates....43
Timer Info....43
About....43
Additional Menus....43
System Tray Menu....43
Generate Password....43
Generate and Show Password....43
Generate and Autotype Password....44
PassCube Password Manager....45
Database File Handling....46
Create or open....46
Save....46
Close....46
Lock/Unlock....46
Export to other file formats....46
Working with the Database....47
Add new entry....47
Edit options....48
Rearranging entries....49
Working with tags....49
View options....49
Searching the database....50
Global database settings....51
Additional functions....52
Configuration File (PwTech.ini)....53
Command Line Options....54
Questions & Answers....55
Which security level is appropriate for my password?....55
Which security measures should I take when generating strong passwords?....56
Is it possible to memorize those random passwords?....57
What about pronounceable passwords?....57
Can I use PwTech as a password safe?....57
Which kinds of word lists does PwTech accept?....58
How to interpret the information about the random pool?....58
Technical Details....60
Random Pool....60
Text Encryption....61
High-Resolution Timer....63
Contact & Further Information....65
Contact....65
Translations....65
Word Lists and Trigram Files....65
Please Donate!....65
Acknowledgment....66

Introduction

The usage of a *password* is still the simplest way to control the access to specific resources. Although many other authentication factors have been developed—examples include identification cards, fingerprint or retinal patterns, voice recognition and other biometric identifiers—, password authentication systems are easier to implement for most applications, are relatively hard to break (note the term "*relatively*"!) and can thus provide accurate security, if used carefully. However, it is essential for the security that the password is *strictly kept secret*, and that it is chosen in a way that makes it hard for an attacker to guess it or to find it by try-and-error (also known as "brute force") or by using dictionaries of common passwords. Both conditions are closely connected, but in a rather fatal way: Passwords which are easy to memorize for humans are for the most part *disastrous* in terms of security! Among these bad examples we find personal data (names of family members, pets, meaningful places, etc.), names and characters from favorite books, films or video games, simple words or character sequences (such as the famous "qwerty"), and so on. These passwords are for sure easy to memorize—but can often be guessed without much effort. How can we solve this dilemma?

There are many ways to choose good (i.e., *secure*) passwords—but the best way is to let a random generator choose a password. If these passwords are long enough, it will take years, if not centuries, to find them by "brute force". Computer programs like PwTech can assist you in generating random passwords, as humans are not very good at making up random numbers themselves. Unfortunately, random character sequences like zio5FcV7J are fairly hard to memorize (although this is possible and probably not as difficult as you might imagine), so you may want to try pass*phrases* composed of words from a word list instead: Five words from a word list with 8000 words or more are sufficient in most cases to create a high-quality passphrase; moreover, the security can easily be increased by adding some random characters.

The need for secure passwords has grown since the advent of the Internet and its many websites where the access to certain resources (message board, user account, and so on) is controlled by a user name/password pair. Fortunately, since the invention of so-called *password safes*, you don't have to remember all these passwords anymore—you just store them in the password safe which is protected by a "master password" (which must be memorized carefully, of course). As this master password is used to protect highly sensitive data, it should conform to the highest security level possible. The security level, which grows with increasing password length, is only limited by the user's ability to memorize random characters or words. With some effort, most people are certainly able to memorize a 90-bit password.

PwTech is capable of generating cryptographically secure random passwords and passphrases conforming to highest security levels. It can be used to generate master passwords, account passwords, and generally all sorts of random sequences—even large amounts of sequences at once. It offers additional useful features such as a password manager/safe, a password "hasher", text encryption, and much more.

Program Features

- Password generation based on a **cryptographically secure pseudo-random number generator** (combination of SHA-256 and AES or ChaCha20)
- **Entropy gathering** by collecting volatile system parameters and measuring time intervals between user keystrokes, mouse movements, and mouse clicks
- **Password manager** functionality through databases encrypted with a master password, containing passwords associated with a title, user name, URL, etc.
- Generation of **passphrases** composed of words from a word list
- **Pattern-based password generation** (formatted passwords) provides nearly endless possibilities to customize passwords to the user's needs
- **Scripting** functionality (Lua) with a programming interface to PwTech allows full control over password generation
- Generation of **phonetic (pronounceable) passwords** based on language-specific trigram (3-letter) frequencies
- Numerous **password options** for various purposes
- Generation of **large amounts of passwords** at once
- **"Password hasher"** functionality
- Secure **text encryption** (AES with 256-bit key)
- **Multilingual support**
- **Full Unicode Support**
- Runs on Microsoft Windows versions beginning with Windows XP, including Windows Vista, 7, 8, and 10 (32-bit & 64-bit)

Unicode Support

PwTech provides full Unicode support as of version 2.3.0. The Unicode standard can—theoretically—encode up to 1,114,112 characters, and the latest version contains a repertoire of more than 110,000 characters. PwTech provides full support for the complete set of Unicode characters, especially in passwords, text encryption, file names, and translations of the program.

Keep in mind that you also need a suitable font to display Unicode characters. TrueType and OpenType fonts can contain up to 65,536 characters, but most fonts on Windows contain only a subset of the first 65,536 Unicode characters. You can use Windows' character map utility (*charmap.exe*) to evaluate the fonts on your system with respect to the characters contained in the font files.

Supported Encodings

Like Microsoft Windows, PwTech uses *UTF-16* (little-endian) as the default character encoding internally. UTF-16 encodes the most frequent characters in 16-bit units, but the number range $1–2^{16}$ (65,536) is actually not sufficient to encode *all* possible 1,114,112 Unicode characters, so some characters have to be encoded as 2x16-bit units. When reading or writing Unicode text from/to files, however, PwTech supports further encodings besides UTF-16 little-endian, namely *UTF-16 big-endian*, *UTF-8*, and *ANSI*.

In UTF-16 big-endian, the byte order is simply reversed compared to UTF-16 little-endian. UTF-8 is an 8-bit variable-width encoding, which means that each character is encoded as 1 to 4 bytes (8-bit units or "octets"). UTF-8 has the advantage that the first 128 Unicode characters, which are encoded as 1 byte in UTF-8, correspond exactly to the 7-bit ASCII character set, thus making ASCII texts valid UTF8-encoded Unicode, and vice versa (for the first 128 Unicode characters).

ANSI is a non-Unicode 8-bit fixed-width encoding which extends the 7-bit ASCII standard (characters 1–128) by an additional set of 128 language-specific characters (characters 129–256). This additional character set depends entirely upon the user's codepage setting in Windows, so ANSI-encoded texts containing non-Latin characters such as *ö, é, î, Å*, etc., written on a machine with a *Western* codepage setting, looks quite different on a computer with a *Greek* codepage setting: The "special characters" would be replaced by characters from the Greek alphabet in this case, simply because the character sets which are used to display the binary codes in the range 129–256 are different on both machines! This cannot happen with Unicode-compliant encodings, since each valid binary code maps to a fixed Unicode character. As a consequence, Unicode-encoded texts look the same on all computers, irrespective of any language-dependent codepage settings in the operating system.

PwTech can identify files containing Unicode text (either UTF-16 little-/big-endian- or UTF-8-encoded) by the so-called "byte-order mark" (BOM) which is a 2-byte (UTF-16) or 3-byte (UTF-8) sequence right at the beginning of the file. If no BOM is present, PwTech assumes the file to be ANSI-encoded. PwTech is also capable of writing UTF-16- and UTF-8-encoded Unicode text files, and conversion of Unicode to ANSI characters is supported, too. The user can change the default file encoding in the *Configuration* dialog on the *Files* page.

Now which encoding should you use? Well, the answer to this question depends on your language and your needs:

- **ANSI**: Using this encoding for texts containing ASCII characters exclusively is unproblematic. However, if the text contains "special" language-specific characters such as *ä, ô*, etc., you may run into trouble if the text file is read on machines with different codepage settings. Thus, ANSI encoding should only be used for Latin-based alphabets, and when the text file is read on computers with the same language settings.

- **UTF-16** and **UTF-8**: The choice between UTF-16 and UTF-8 largely depends on the character set of the text to be encoded, and also on the target application of the encoded Unicode text. UTF-8 is more efficient with respect to file size for Latin-based alphabets, whereas the alphabets of Greek, Cyrillic, Hebrew, Arabic, Coptic, Armenian, Syriac, Tāna and N'Ko require 16 bits in both UTF-16 and UTF-8. The rest of the characters of most of the world's living languages is more efficiently encoded as UTF-16 (16 bits needed) compared of UTF-8 (24 bits needed). Furthermore, in Windows, which internally uses UTF-16, reading and processing UTF-8-encoded files may be (slightly) slower in some cases because the encoding has to be converted to UTF-16 before passing the text to Windows controls. On the other hand, UTF-8 is the *de-facto* standard encoding of the Internet, and should therefore be preferred for Internet-associated document types.

- **UTF-16 little-endian** *vs.* **big-endian**: UTF-16 little-endian should be preferred on the Windows platform.

Happy unicoding!

Password Generation with Password Tech – An Overview

PwTech will assist you in generating cryptographically-strong and easy-to-memorize passwords and passphrases. The "design philosophy" behind PwTech is to provide a simple-to-use and unobtrusive, yet at the same time powerful and versatile application intended at professionals (e.g., system administrators) *and* "normal" home users.

PwTech allows you to generate various types of passwords and passphrases:

- **"Classical" passwords** such as `VLyw68JsSq0m`, i.e., random sequences of characters from a specific character set (upper-case/lower-case letters and numbers in the example).

- **Phonetic (pronounceable) passwords** such as `terweeptoton`, which are generated by evaluating the frequencies of all possible trigrams (1 trigram = 3-letter sequence; for 1 trigram, there are 26^3 = 17,576 possible letter combinations) of a certain language (English in the example).

- **Passphrases** such as `khaki cello waxy mecca verdi`, composed of randomly chosen words from a word list. Words may easily be combined with random characters to generate passphrases such as `lends-ah susie-Tx chats-Hz joins-TE`.

- **Formatted passwords / passwords based on patterns**: This is the most versatile password generation feature, for it provides a variety of format specifiers to insert characters from pre-defined character sets and words from a specific word list into the resulting password. Moreover, it allows for repeating and randomly permuting sequences in the password. *Example*: The format string `{4u4l2ds}` means "*Insert 4 upper-case letters, 4 lower-case letters, 2 digits, and 1 special symbol; permute the character sequence afterwards*", and yields passwords such as `fP4eM#mAiV2`.

- **"Scripted" passwords**: PwTech provides a programming interface for Lua scripts, which allow almost full control over the process of password generation and nearly endless options to customize passwords, passphrases, and any other kind of random keys.

- **"Hashed" passwords**, passwords based on a master password and a parameter: Similar to the website *Hashapass*, this feature enables you to reproducibly generate unique passwords using a (secret) master password and a (not necessarily secret) parameter, such as the name of a website. For example, the master password "qwerty" (*hint: don't use that one, ever!*) together with the parameter "yahoo" yields the password `W4h-f0L21K3FYb8Qx`. PwTech's password hasher offers several pre-defined character sets for the user to choose from. Being independent of any encrypted databases of stored parameter–password combinations, it may thus function as a fully portable "password safe". In case you want to be independent of Windows-based applications, PwTech can also provide full compatibility with the *Hashapass* generator, which is accessible via the Internet and should work with any browser.

To give the user an estimation of the quality of the generated passwords, PwTech displays a "password quality bar" featuring a color range from red-orange (less secure) to dark green (more secure). A password quality/security—or, in more technical terms, *entropy*—of 128 bits is considered unbreakable by today's computer technology, and, given the huge complexity of the 128-bit key space ($\sim 3.4 \cdot 10^{38}$), I really think one can be quite certain that 128-bit passwords and keys will remain secure for the entire era of humankind on planet earth.

Step-by-Step Tutorial

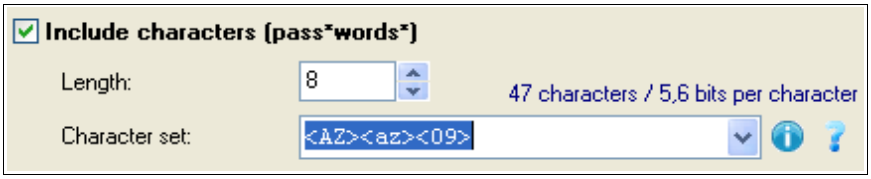
The following sections describe the key functions and parameters of the program for generating various types of passwords. They are readily available in PwTech's main window.

Profile Selection



Select a profile from the drop-down list to load a custom password generation profile. Note that, after loading a profile, changing individual password settings does not automatically update the selected profile, but the profile selection still remains active. To reload the selected profile, press the **button**. To add a new profile, press the **button**, which opens the *Profile Editor* where you can add a new profile or overwrite an existing one.

Include Characters



Check **Include characters** to include characters in your password. Enter the desired length of the character string in the **Length** field (maximum 10,000).

Enter the desired character set (i.e., all the characters that may occur in your password) in the field below or choose one from the drop-down list. Note that, in a *set*, each character must occur only once. PwTech aims at providing full Unicode support as of version 2.3.0, so you can (theoretically) enter up to 1,114,112 different characters. You can use Windows' character map utility (*charmap.exe*) to insert Unicode characters which are not accessible via the keyboard.

Furthermore, you may use the following placeholders to abbreviate the sequence (some placeholders may be further abbreviated to save space in the input box):

Placeholder(s)	Meaning	Character set
<AZ>	Upper-case letters	ABCDEFGHIJKLMNOPQRSTUVWXYZ
<az>	Lower-case letters	abcdefghijklmnopqrstuvwxyz
<09>	Digits	0123456789
<Hex>/ <HEX>	Upper-case hexadecimal symbols	0123456789ABCDEF
<hex>	Lower-case hexadecimal symbols	0123456789abcdef
<base64>/ <b64>	Base64 symbols	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
<easytoread>/ <etr>	Like *<AZ><az><09>*, but without similar-looking ("ambiguous") characters (B8G6I1l|0OQDS5Z2 by default). See *Advanced Password Op-*	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 *without ambiguous characters*

	tions for more options.	
<symbols>/ <sym>	Additional symbols accessible via the keyboard	`!"#$%&'()*+,-./:;<=>? @[\]^_`{|}~`
<brackets>/ <brac>	Brackets	`()[]{}<>`
<punctuation>/ <punct>	Punctuation marks	`,.;:`
<highansi>/ <high>	Higher ANSI characters (symbols #127 to #255)	*(characters depend on the current ANSI code page)*
<phonetic>	Generate phonetic (pronounceable) passwords based on trigram frequencies of the English (or a user-specified) language *(see notes below)*	<*phonetic*>: lower-case letters <*phoneticu*>: upper-case letters <*phoneticx*>: mixed-case letters *(frequencies of the letters are language-dependent; English by default)*
<phoneticu>		
<phoneticx>		
<XY>	Add characters in the Unicode range from *X* to *Y* (inclusive)	*variable, max. 256 characters*

To add a range of Unicode characters to the character set, you can use the parametric placeholder ***<XY>***, with *X* and *Y* being the start and end Unicode character, respectively. The code point corresponding to *Y* must be higher than that of *X*, and the difference between the code points must not be higher than 256. Thus, a maximum of 256 Unicode characters can be added to the set with this placeholder. For example, the placeholder < ~> adds 95 characters ranging from code point #32 (U+0020, space character) to #126 (U+007E, tilde character). *Note: The code point corresponding to Y must not be higher than U+7DFF.*

You may provide a **comment** included in square brackets [...] at the beginning of the entry. All characters inside the comment will be ignored by PwTech. For example, entering `[This is a comment.]<AZ>` will add only upper-case letters to the character set. Note that the comment must be placed right at the beginning; otherwise, it will be treated as a normal sequence of characters.

If the text in the box is equal to `<phonetic>`, `<phoneticu>`, or `<phoneticx>` (other characters are *not* allowed!) PwTech will apply a special algorithm to generate **phonetic passwords**, i.e., passwords that are likely to be pronounceable. The `<phonetic>` and `<phoneticu>` versions generates passwords exclusively composed of lower-case and upper-case letters, respectively, whereas passwords generated with the `<phoneticx>` version can contain both lower-case and upper-case (i.e., mixed-case) letters. The algorithm is based on the language-specific frequencies of so-called *trigrams*, which consist of three letters from the English alphabet (a–z, 26 symbols). Phonetic passwords look like this: `rationeterbonte`. You can also generate phonetic passwords based upon another language by loading a "trigram file" containing the trigram frequencies that are characteristic of that language (note, however, that this is only possible for languages the alphabets of which are Latin-derived!). For example, using the trigram frequencies stored in the file *German.tgm* yields passwords such as `gergenfortenman`. To load a trigram file, you have to modify the *Advanced Password Options*. You may also create a trigram file yourself by evaluating a dictionary, word list, or any other text of your choice (see *Tools* | *Create Trigram File (F6)* in the main menu).

The algorithm for generating phonetic passwords is sensitive to the options "*Include at least ...*" (see: *Advanced Password Options*), so you may include an upper-case character, digit, and/or special symbol in the resulting password in order to increase the security. It is, however, not possible to combine the phonetic rule with other character sets (digits, symbols, …) because this would destroy the "pronounceability" of the sequence. Alternatively, you can use the placeholder q in a format sequence and combine it with any other character set (see: *Format Password*).

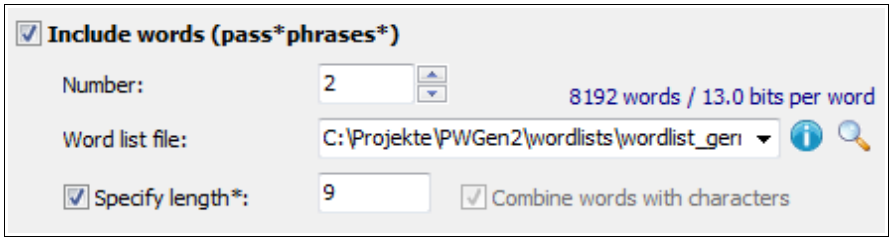
The character set will be updated as soon as you leave the input field, e.g., by clicking on another interaction element. The entropy (number of bits per character) is displayed on top of the *Character set* field: For "normal" (non-phonetic) passwords, it is calculated as $\log_2 N$, where N is the number of characters and $\log_2$ is the logarithm base 2; for phonetic passwords, the entropy is lower than $\log_2 26$ because the letters all have different frequencies (~3.6 bits per character for the default trigrams).

Clicking on the **symbol** shows a quick help box with a description of the available placeholders.

Clicking on the **symbol** shows a message box with the currently loaded character set.

Errors occur when the set contains less than two unique characters. Keep that in mind if you want to generally exclude ambiguous characters (see below).

Include Words



Check **Include words** if you want to include words from a word list in your password. Enter the desired number of words into the **Number** field (maximum 100).

By default, PwTech uses an internal English word list containing 8192 (2^{13}) words. It is activated by entering `<default>` into the **Word list file** field or by simply leaving this field blank. To load a custom file, enter its name into the field or press the **symbol** to browse through your folders and select a file. Note that PwTech supports Unicode word lists and is capable of reading Unicode text files encoded as UTF-16 or UTF-8, provided that the file contains the corresponding byte-order mark (BOM) at the beginning of the file. If the BOM is missing, the file is assumed to be ANSI-encoded.

Comments may be provided in square brackets […] before the actual file name, for example: `[My own wordlist]C:\Users\user\Documents\my_own_list.txt`. Comments at the beginning of entries may help you find specific word lists with long file names in the list.

The word list will be loaded as soon as you leave the input field, e.g., by clicking on another interaction element.

To **reload a word list** from an already loaded file, load the default list, then load the desired file again by selecting it from the drop-down list.

By default, PwTech accepts words that have at most 30 (Unicode) characters; however, you can reduce this limit in the *Advanced Password Options* dialog. The number of bits per word is calculated as $\log_2 N$ (with N being the number of words) and will be displayed on top of the *Word list file* field. The size of word lists is limited to 1,048,576 (2^{20}) words.

Clicking on the **symbol** shows information about the currently loaded word list.

Errors occur when the word list cannot be opened, or when it contains less than two valid words.

Select **Combine words with characters** to generate passwords as a combination of words with characters. This means that each word will be combined with one or more characters, depending on the number of words and the number of characters, respectively. Example of 3 words combined with 8 characters by a "-" symbol: `putty-fuV umbra-Sxq faint-fT`.

Select **Specify length** to specify a length range for the passphrases in the edit box on the right. This option may be helpful for word lists in which the words exhibit a wide range of lengths, as in case of the default English word list. If activated, only passphrases with lengths conforming to the specified length range are allowed, whereas non-conforming passphrases will be discarded.

By default, the length specification refers to the *characters from the actual words and space characters between the words*, without counting characters from a character set via the *Include characters* option (if applicable). To consider *all* characters in the resulting passphrase, insert an asterisk (*) somewhere in the sequence, e.g., "*15-20" or ">10*".

Depending on the length specification, it may take some time until a randomly generated passphrase finally meets the length criterion; in the worst case, there is no passphrase combination at all that meets the criterion. When the number of failed attempts reaches an internal threshold, a progress window with a *Cancel* button will be shown, which allows the user to cancel the process.

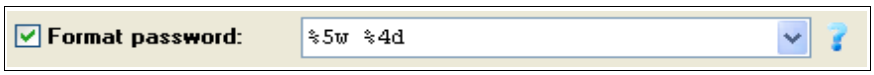
The length can be specified in various **formats**:

Format (*M*,*N* = number)	Meaning
N	Length must be exactly *N* (characters)
M-N	Length must be at least *M* and at most *N*
>*N*	Length must be greater than *N* (at least *N*+1)
>=*N*	Length must be at least *N*
<*N*	Length must be less than *N* (at most *N*−1)
<=*N*	Length must be at most *N*
* *(specified anywhere in the sequence)*	Length specification refers to *all* characters in the passphrase (words *and* characters from a character set, if any)

Note that a length of 0 is not allowed. PwTech does *not* check the validity of the specified length range for the current password/passphrase settings. It is the user's responsibility to provide a length range that is compatible with the chosen settings.

The label of the checkbox is marked with an asterisk (*) to indicate that this option can decrease the password security. The extent of this reduction depends on the width of the restriction in most cases: The shorter the passphrase and the narrower the length range, the greater the potential decrease in security. Yet filtering out short passphrases may actually be beneficial overall because short passphrases, when treated as pass*words* (i.e., sequences of individual characters), tend to be insecure compared to long passphrases. Thus using a length specification such as "*>N*" (see table above) can effectively increase the security by avoiding short and insecure passphrases.

Format Password



Select **Format Password** to format your password or passphrase generated via *Include Characters* and/or *Include Words*. Additionally, you can use format specifiers and placeholders to introduce random characters from various character sets, and random words from the word list.

Format strings may contain two types of characters, **format specifiers** and **literal characters.** Literal characters are copied verbatim to the resulting password and must be enclosed in quotation marks "...". Format specifiers can be used to insert random characters and words, or to manipulate the input sequence (e.g., by randomly permuting a character sequence). A literal quotation mark may be inserted by entering double quotation marks.

Format specifiers have the following form: ***[*][number](s***. Arguments enclosed with square brackets [...] are optional, those enclosed with brackets (...) are required.

- The optional ***number*** argument indicates how many times the format command is to be repeated; it can be a decimal number in the range from 1 to (theoretically) 99,999. If *number* is not specified, "1" is assumed by default. Alternatively, this argument can also be specified as a **number range** in the form *N1-N2*; then, a randomly chosen numberbetween *N1* and *N2* will be inserted.

- The ***specifie***

r
argument is the actual format specifier or placeholder for a random character from a certain character set (see below for a list of format specifiers) or for a random word from the currently loaded word list.

- The optional ***as t e r- is k*** **(*)** argument instructs the program to use each character/ word in the following sequence only once. In this case, the number of characters/ words is limited to the size of the character set or word list. This argument can be placed *anywhere* between the percent sign and the format specifier. Note that if the asterisk is specified but the *number* argument is omitted, the size of the character set or of the word list, respectively (depending on the format specifier), will be used as the default number.

Examples:

- u means "*Insert 1 random upper-case letter*".
- 20a means "*Insert 20 random lower-case alphanumeric characters*".
- *5-10d means "*Insert between 5 and 10 (exact number chosen randomly) random digits, each digit must occur only once in the sequence*".
- *d means "*Generate permutation of digits 0-9*".

The **length of formatted passwords** is currently limited to 16,000 (Unicode) characters.

Some format specifiers have **special meanings.** In the following list, "*[N]"* means that the *number* argument may be specified optionally:

- **P**: Insert the password generated via *Include characters* and/or *Include words*. The password can be inserted only once! A "warning" message will be displayed on top of the input field, if ...
  - *Include characters* or *Include words* is checked, but **P** is not specified in the format string ("P is not specified.");
  - **P** is specified, but neither *Include characters* nor *Include words* is checked ("P: password not available.");
  - The password is too long to insert it at full length, i.e., inserting the full password would exceed the size limitation of the formatted password ("P: password too long.").

  Example: `"This is your password:" ""P"".`

Output: *This is your password: “Zq3wu9gL”.*

- ***[N]*`w` or ***[N]*`W`**: Insert random word(s) from the currently loaded word list (see above). In case of multiple words (*N* > 1), the words are separated by a space if the specifier is **w** (lower-case), or they are not separated at all if the specifier is **W** (upper-case). Note that *Include words* doesn’t have to be checked in order to use the word list for formatted passwords.

  Example: `5w`
  dod zion belt xylem avery

- ***[N]*`[...]`**: Repeat the sequence included in the brackets **[** and **]** *N* times. Repeat sequences may be nested up to 4 times (e.g., `5[...4[...3[...2[...] ] ] ]`).

  Example: `5[ad]`
  u9o2p9r2a8

- ***[N]*`{...}`**: Randomly permute (shuffle) the sequence included in the brackets **{** and **}** and insert *N* characters from the permuted sequence. Nesting is not possible (because it doesn’t make any sense). Please note that, when PwTech calculates the quality/security of the resulting password, it does *not* take into account the security gain caused by the random permutation, which is very likely unless the permuted sequence consists of characters from only one single character set (in the latter case, there wouldn’t be any increase in security). An exact calculation of this security gain is not trivial and can be computationally expensive, so this feature is currently not available. However, you can try to calculate the gained security bits by this formula:

  $S = \log_2(N! / (n_1! \, n_2! \, \ldots \, n_i!))$, where N is the total number of characters in the permuted sequence, and n_i are the numbers of characters from a specific character set.

  Examples: (1) `{“hello world!”}`
  ldloor w!lhe
  (2) `{9ld}`
  euvk8bfifg

- ***[N]*`<<...>>`**: Treat the sequence enclosed with the brackets << and >> as a character set, and choose *N* random characters from this set. Note that within this sequence, format specifiers don’t have any effect (except for >> which ends the sequence); each character will be used “as-is”. It is possible to use abbreviation codes, i.e., placeholders for character sets (see *Include Characters*), such as `<AZ>`, `<09>`, and so on.

  Example: `<<<az>0123%+/>>`
  r0/bpya%0y

Comments enclosed with square brackets [...] may be provided at the beginning of the sequence (and only there!).

The complete list of format specifiers which are currently supported by PwTech is shown in the following table:

Format specifier/ placeholder	Meaning	Character set
1) Placeholders for random characters from various character sets (asterisk and number argument may be specified optionally in all cases)		
x	insert random character(s) from the user-defined character set (as given in the input field below *Include characters*)	*(user-defined)*
a	lower-case alphanumeric	`abcdefghijklmnopqrstuvwxyz0123456789`
A	mixed-case alphanumeric	`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789`
U	upper-case alphanumeric	`ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789`
E	mixed-case alphanumeric, but without ambiguous characters	`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789` *without ambiguous characters*
d	digit	`0123456789`
h	lower-case hexadecimal	`0123456789abcdef`
H	upper-case hexadecimal	`0123456789ABCDEF`
l	lower-case letter	`abcdefghijklmnopqrstuvwxyz`
L	mixed-case letter	`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz`
u	upper-case letter	`ABCDEFGHIJKLMNOPQRSTUVWXYZ`
v	lower-case vowel	`aeiou`
V	mixed-case vowel	`AEIOUaeiou`
Z	upper-case vowel	`AEIOU`
c	lower-case consonant	`bcdfghjklmnpqrstvwxyz`
C	mixed-case consonant	`BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz`
z	upper-case consonant	`BCDFGHJKLMNPQRSTVWXYZ`
p	punctuation marks	`,.;:`
b	brackets	`()[]{}<>`
s	special symbols (may be user-defined)	`!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~` *(or user-defined)*
S	mixed-case alphanumeric and special symbols	`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!"#$%&'()*+,-./:;<= >?@[\]^_`{|}~` *(special symbols may be user-defined)*
y	higher ANSI characters (symbols #127 to #255)	*(characters depend on the current ANSI code page)*
q	generate phonetic password composed of lower-case letters exclusively	`abcdefghijklmnopqrstuvwxyz` *(frequencies of the letters are language-dependent; English by default)*
Q	generate phonetic passwords	`ABCDEFGHIJKLMNOPQRSTUVWXYZ`

	composed of upper-case letters exclusively	*(see notes above)*

r	generate phonetic passwords composed of mixed-case letters	`abcdefghijklmnopqrstuvwxyz`
		`ABCDEFGHIJKLMNOPQRSTUVWXYZ`
		(see notes above)

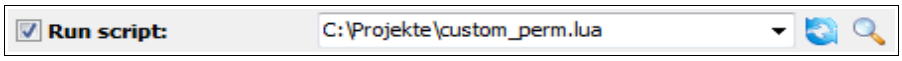
2) Special format specifiers (optional arguments indicated in *italics*)		
P	insert password generated via *Include characters* and/or *Include words*; password may be inserted only once	
***[*][N]*w**	word from word list; multiple words are separated by a space	
***[*][N]*W**	word from word list; multiple words are concatenated without any separators	
***[N]*[**	repeat input sequence included in the brackets *N* times; maximum nesting depth is 4	
]		
***[N]*{**	randomly permute formatted sequence included in the brackets and keep *N* characters from the permuted sequence	
}		
***[N]*<<**	treat the character sequence included in the brackets as a character set and insert *N* characters from this set	
>>		

With the format option at hand, you can easily define your own **rules for creating passwords for specific purposes**. Some examples are given in the following table.

Rule	Format sequence
4 upper-case letters, 4 lower-case letters, 2 digits, 1 special symbol *in random order*	`{4u4l2ds}`
8 alphanumeric characters, of which at least 1 is a letter and at least 1 is a digit	`{6ALd}`
4 artificial words, where each word consists of 6 letters and is composed of alternating consonants and vocals	`4[3[cv] ]`
3 artificial “phonetic” words (generation is based on trigram frequencies) with 8 letters each	`3[8q ]`
Phonetic password interspersed with digits and special symbols	`6q2ds6q2ds`
5 words from the currently loaded word list, each combined with 2 digits	`5[w-2d ]`
Random permutation of upper-case letters	`*u`
Random number in the range 0123-9876, each digit must occur only once	`*4d`
10 alphanumeric characters, first character must not be a lower-case letter	`U9A`
Random product ID	`5d-”OEM”-7d-5d`
Hexadecimal 128-bit key (e.g., WEP key)	`32h`
Random MAC address (48-bit)	`5[2h-]2h`
12 Random base64 symbols	`12<<<base64>>>`
128-bit key in binary notation	`128<<01>>`
8 random characters from the German alphabet	`8<<<AZ><az>ÄÖÜäöüß%>>`

3 artificial *German* words, where each word consists of 6 letters and is composed of alternating consonants (including *ß*) and vocals (including *ä*, *ö* and *ü*)	`3[3[<<bcdfghjklmnpqrstvwxyzß>><<aeiouäöü>>] ]`
At least 4, at most 8 words from the currently loaded word list, each word must occur only once	`*4-8w`

Run Script



Select **Run script** to execute a custom *Lua script* for password generation. *Lua* is a high-level, dynamically typed programming language, providing basic mathematical functions, string operations, and list/array functionalities.

PwTech provides a programming interface to the Lua scripting layer, which allows you to …

- Manipulate the password generated via the *Include …* options described previously;
- Generate random numbers, passwords, words, passphrases, and formatted passwords;
- Access various user inputs, including password settings and advanced password options;
- Set a new password or a manipulated input password and the entropy value associated with it.

Enter the name of the script file into the box, or press the **button** to browse folders and select a file. The script is loaded when generating a password, and is kept in memory until the file name is changed. To remove the loaded script from memory, press the **button**.The file is reloaded when generating a password the next time. Just like with the *Format password* option, **comments** enclosed with square brackets […] may be provided before the file name.

Refer to the *PwTech Scripting Manual* (file *scripting.pdf*) for a complete description of the Lua programming interface, which includes functions that are called by PwTech during password generation, and functions that can be called from the scripting layer.

Advanced Password Options



Press the **Advanced** button to open a window that allows you to access various advanced (extended) password options. The **list of checkboxes** on top of the window provides the following options that can be activated or deactivated individually:

- **Exclude ambiguous characters:** Excludes those characters from character sets that might be confused with other similar-looking characters (that is, by default, `B8G6I1l|0OQDS5Z2`). As this option reduces the size of the character set, the password security will be reduced accordingly. This option also applies to formatted passwords and affects all character sets encoded by the placeholders a, A, U, etc. This option does not affect phonetic passwords.

- **First character must not be a lower-case letter:** Activate this option if you don't want the first password character to be a lower-case letter (a–z). This might be useful when copying passwords to certain word processors or e-mail programs that automatically convert the first character to upper-case in case it is a lower-case letter, thereby manipulating the original password. If the first character belongs to a word, it will simply be converted to upper-case. If it is, however, a random character, PwTech will choose a character that is *not* a lower-case letter (for example, if the character set consists of lower-case letters and numbers, the first character will be a number). If the character set does not contain any non-lower-case letters, the first character will be an upper-case letter. Note that activating this option (slightly) reduces the password security.

- **Don't separate words by a space:** By default, PwTech inserts spaces to separate words in passphrases. Select this option to deactivate this behavior.

- **Don't separate words and characters by a '-' character:** By default, PwTech inserts a minus sign ("`-`") between a word and a character (if the option *Combine words with characters* is selected). Select this option to deactivate this behavior.

- **Reverse default order of character/word combinations:** When you generate passwords/passphrases consisting of both characters and words, PwTech by default puts the characters at the beginning and appends the words. If *Combine words with characters* in the main window is activated, a subset of the character sequence is appended to each word (e.g., `word1-3wp word2-k8#`). If *Reverse default order …* is activated, however, the order is reversed in each case: If *Combine ...* is activated, each word is appended to a subset of characters (e.g., `3wp-word1 k8#-word2`); otherwise, the words are put first, and the characters are appended.

- **Include at least …:** Activate these options to include at least one character from the given character sets, upper-case letters (A–Z), lower-case letters (a–z), digits (0–9), and special symbols (`!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`, or user-defined symbols, see below). If the user-specified character set does not contain any characters from a predefined set, the resulting passwords will contain exactly one character from this predefined set. Otherwise, they will contain *at least* one character from this set. For example, if you specify a <az> character set and activate all of the aforementioned options, PwTech will generate passwords like this one: `t]cXy7cu`. (If the desired password length is less than 4, not all characters can be included, of course.) Depending on the user-specified character set, selecting these options can—rather slightly—reduce (but in some cases also *increase*) the password security. Note that these options do not affect the generation of pass*phrases* or formatted passwords.

- **Only include characters from custom character set:** This option refers to the *Include at least …* options described above. If activated, characters from the user-defined (custom) character set will be used exclusively for generating passwords. For example, if the character set is defined as `abcdEFGH012345+/` and all four *Include at least …* options are activated, generated passwords will contain ≥1 lower-case letter from the set `abcd`, ≥1 upper-case letter from the set `EFGH`, ≥1 digit from the set `012345`, and ≥1 special symbol from the set `+/`. If the custom set does not contain any matching char-

acters from a given *Include …* character set, characters from that set will not be included in the resulting password. This would be the case for the custom character set `abcdEFGH012345` and the option *Include at least one special symbol*: As the custom set does not contain any special symbols, generated passwords will not contain any special symbols either. Also, note that the "special symbols" character set may be redefined (see below), which would also affect this option with respect to matching characters in the custom character set. The option has no effect on phonetic passwords, i.e., these passwords always contain characters from the default *Include …* character sets if the corresponding options are selected.

- **Exclude repeating consecutive characters:** Choose this option if you want to avoid repeating sequences of the same character, as in `r3`**`aa`**`5ZiK` or `E0u`**`555`**`Wp`, for example. If this option is activated, each character in the sequence will be different from the previous one. Note that this restriction reduces the security of the passwords, but the extent of the reduction strongly depends on the size of the user-defined character set: For large character sets, the effect becomes almost negligible. For example, if your set contains 64 characters (6 bits per character) and the password length is 16, the loss of entropy is only 0.34 bits of 96 bits (0.35%). The effect is more pronounced for smaller character sets: If your set contains only 4 characters (2 bits per character) and the password length is 48, the loss of entropy is 19.5 bits of 96 bits (20.3%). This option also applies to formatted passwords.

- **Exclude duplicate entries in password lists:** Select this option if you want to generate "unique" password lists where each entry occurs only once. Generating unique lists may take longer than usual due to the extra check whether the entries to be added already exist in the list. It also requires roughly the double amount of memory. In certain cases, it might not be possible to generate complete password lists using the parameters given by the user; for example, if the user chooses `0123456789` as the character set (= 10 possibilities) and 3 as the password length, it is not possible to generate a list of more than 1000 (= 10^3) unique entries, since the total number of possible password permutations is limited to this exact number. As a consequence, the security of the entire password list is decreased, since the number of possibilities decreases from top to bottom in the list. However, when a single password independent of the rest of the list is considered, it retains its designated security.

- **Convert all words in word lists to lower-case:** When loading word lists, each word in the list must occur only once, and the words are compared in a case-sensitive manner. So if you do not want the list to contain both upper-case and lower-case words, you can activate this option and thus force PwTech to convert each letter in a word to lower-case.

- **Each character/word must occur only once:** With this option, each character or word will occur only once in passwords or passphrases, respectively. Note that this option limits the password length and the number of words to the size of the character set and of the word list, respectively. If you enter a number exceeding this limit, you will receive a warning message, and the number will be decreased automatically. Also note that this option *always* decreases the password security: For example, if the size of the character set is given by *N*, and *M* characters are to be inserted from this set, the en-

tropy of the resulting sequence can be calculated according to $S = M \log_2 N$ or $S_{red} = \log_2(N!/(N-M)!)$ if this option is *de*activated or activated, respectively. Due to the factorial terms S_{red} is always smaller than S (for $M > 1$). The effect of entropy reduction becomes less pronounced with increasing N and comparably small M ($N >> M$): For $N = 26$ and $M = 12$ (e.g., 12 random letters), S is 56.4, whereas S_{red} is 52.0, corresponding to an entropy reduction of 8%. However, for $N = 8192$ and $M = 12$ (e.g., 12 random words from the standard word list), the effect is practically negligible (156 vs. 155.99, corresponding to a reduction of 0.006%). When this option is activated or the "asterisk" (*) argument in formatted passwords (see: *Format Password*) is used, PwTech calculates the reduced entropy S_{red} for the resulting password.

Most of these options can potentially—but not necessarily!—reduce the security of the resulting passwords. See the corresponding notes for the options above for more details. If one or more of these options are activated, the caption of the *Advanced* button is marked with "(!)".

Tip: Right-clicking on the list of options opens a menu where you can (de)select all items or invert the current selection.

Redefine ambiguous characters: The set of ambiguous characters is defined as `B8G6I1l| 0OQDS5Z2` by default. Redefining this set also affects the other pre-defined character sets (i.e., *<AZ>, <az>, <09>,* etc., as well as all the character sets accessible via the *Format Password* option). You can also supply this character set in *groups* of similar-looking characters, separated by a space. Then, each group of this set is only excluded from the user-defined character sets if they contain *two or more* characters from this group; if only one character is used, the group will *not* be excluded. Please note that

- A group must consist of at least two characters.
- Each character must only occur once in the whole sequence, i.e., it cannot be part of more than one group.
- You can still exclude the space character itself by putting it at position 1 or 2 in the sequence (e.g., `1 0OQD` is not a valid definition of groups, so the sequence will be treated as a "normal" set of ambiguous characters, including the space character).
- Group definitions exclusively apply to user-defined character sets (i.e., via option *Include Characters* and via format specifier <<...>>). For all pre-defined character sets, *all ambiguous characters* present in the sequence will be excluded, independent from any group definitions!

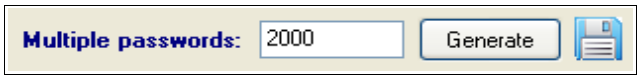
Example: Enter `B8 G6 I1l| 0OQD S5 Z2` to define *six groups* of similar-looking characters. If your character set used for password generation is defined as <az><09> (i.e., lower-case letters and digits), only the characters `1` and `l` from group 3 will be excluded, whereas all the characters from the other groups remain unaffected, because your character set doesn't contain any other characters from the groups which might be confused with the characters in the set. However, if you add upper-case letters to the set, all six groups will be excluded.

Redefine special symbols: The set of special symbols is defined as `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~` by default. Redefining this character set affects the <*symbols*> character set, the option *Include at least one special symbol,* and the corresponding placeholder character sets in the *Format Password* option.

The field **Maximum length of words in word lists** should be self-explanatory. When loading a word list, only those words the lengths of which do not exceed this value will be added to the list. The maximum word length can range from 1 to 30 (default). Changing this value is particularly useful for filtering very long words, or to reduce the size of the entire word list. Note that this option does not apply to the default word list but only to lists from an external source.

Trigram file for generating phonetic (pronounceable) passwords: Here you can specify a special "trigram file", i.e., a file containing the frequencies of all 17,576 (26^3) possible trigrams (= 3-letter combinations: *aaa, aab, …, zzz*) for a given language. This file should have the extension ".tgm". PwTech uses the trigram frequencies to generate phonetic passwords (see *Include Characters*). Trigram files can be downloaded from the PwTech download page, but you may also create your own files via the menu item *Tools* | *Create Trigram File (F6)* in the main menu. If the input field is left blank, PwTech will use its internal (English) trigram table.

Generate Multiple Passwords



If you want to generate more than one password and display the list in a separate window, enter the exact number into this field (maximum 2,000,000,000) and click on **Generate**. The program will open a window showing the password list. Right-click into the window to access the context menu with additional functions:

- **Copy:** Copies the current text selection to the clipboard.
- **Encrypt & Copy:** Encrypts the selected text and copies the ciphertext to the clipboard (see also: *Encrypt/Decrypt Clipboard (F3/F4)*)
- **Select All:** Selects the entire text.
- **Save As File:** Writes the text to a file of your choice.
- **Change Font:** Changes the text font.

You may also **drag & drop** the password list or a selection thereof by left-clicking on a text selection, holding the mouse button, and dragging the text to other applications which have been registered as "drop targets".

Note: The size of the password list is limited to 500,000 bytes in the 32-bit version and 4,000,000,000 bytes in the 64-bit version of PwTech. Generating very large lists may exceed the available memory (of your system or of the program) and cause an "out of memory" error, especially in the 32-bit version, which is limited to 4 GB of memory. If the option *Exclude duplicate entries in password lists* (see: *Advanced Password Options*) is activated, the memory requirement is roughly doubled due to the creation of a binary search tree, which allows for a fast search of the existing password entries.

If you want the list to be **directly written to a file** instead of being displayed in a window, press the **button** and specify a file in which the list is to be stored. PwTech will write each password to this file without any limitations regarding memory requirements. Instead, this

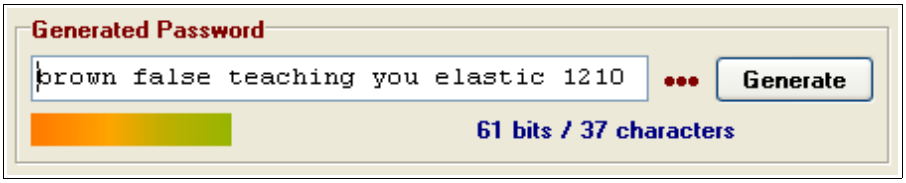
process only requires a negligible amount of memory, and the amount of data that can be generated is only limited by the free disk space on the device. Thus, you can theoretically generate several gigabytes of data (and more … but note that I have not tested this myself!). Note that this low memory requirement does *not* apply to unique lists generated with the option *Exclude duplicate entries*.

PwTech is capable of writing Unicode-compliant text files encoded as UTF-16 or UTF-8 (depending on the corresponding setting under *Files* in the configuration; 8-bit non-Unicode ANSI conversion is supported, too). Each generated password will be encoded accordingly before being written to the file.

If the destination file already exists, you have the possibility of extending the list by appending new passwords to the file. This function is not compatible with the option *Exclude duplicate entries*, i.e., the newly generated passwords will not be checked against the already existing passwords in the file. Instead, a new unique list will be generated and appended to the file.

If you generate a larger amount of passwords (more than, say, 1000 passwords), PwTech shows a progress window with a *Cancel* button. Click on *Cancel* to stop the process and show a list of the passwords which have been generated so far.

Generate Single Passwords



If you want to generate one single password, click on **Generate**. PwTech will display the password in the box.

The estimated security of the password is indicated by the colored "**security bar**" below the password box. The color of this bar ranges from red-orange to light green, i.e., from "*low security*" to "*high security*", where the limit for "*highest security*" has been—more or less arbitrarily—set to 128 bits (which is just some kind of a "magic number" for cryptographers). Always stay on the green side when generating "serious" passwords!

Beware of common (bad) passwords: By default, PwTech tests every password against a list of the 10,000 most common passwords. If the password matches any entry in this list, the password security is decreased to $\log_2(10{,}000) \approx 13$ bits (very low), and the text label next to the security bar is marked with two asterisk symbols (**). *It is strongly recommended to discard these passwords, as they can easily be cracked by password cracking tools using dictionary attacks.*

Drag & Drop: Left-click on the security bar or on the "*Generated Password*" group box (on top of the password box) and hold the mouse button to start a drag & drop operation. Keep the button held down while dragging the password into other applications which have been registered as "drop targets". If the password box contains a selection of characters, only this selection will be dragged & dropped. Otherwise, the entire password will be used. Conversely, you

can also drop text from other applications into the password box, provided that the *Editable* option (available in the context menu of the box) is enabled.

Autotype: Right-click on the password box to open the context menu and select **Perform autotype** to let the program automatically type the password in another application (see *Configuration* | *General* for more details on the autotype feature).

Note that the *security of a single password is limited to 256 bits* because the "random pool" which is used to generate passwords has an effective size of 256 bits (which corresponds to the message-digest length and the key length of SHA-256 and AES/ChaCha20, respectively). However, if the random pool hasn't been filled with enough entropy before password generation, the actual security is lower than the value shown in this field—to be more precise here, the pool must have been filled with *at least N* bits with *N* being the password security. For more information on this topic, see: *How shall I interpret the information about the random pool?*

If—for whatever reason (severe paranoia?)—you want to create a password with a security of more than 256 bits, you may concatenate two or more 256-bit passwords to a 512-bit (or more) password. This is done as follows: First, collect entropy for a 256-bit password, generate it and store it somewhere; then collect entropy for a second password and concatenate that with the first one. Note, however, that there are *absolutely no concerns* that it will *ever* be possible to break 256-bit keys.

PwTech also enables you to **test your own password creations**. In the context menu of the password box, activate the options **Editable**, which allows editing the contents of the text box, as well as **Enable Password Testing**. Now you can type into the box or paste text from the clipboard, and PwTech *estimates*(!) the security/quality (in bits) of the entered sequence. A leading asterisk symbol (*) will be displayed in the label (i.e., "***xx bits / xx characters***"), indicating that the displayed security value in bits is roughly estimated. Note that the underlying algorithm is quite simple and does not make use of any sophisticated statistical tests for randomness (e.g., autocorrelation, runs test, entropy estimation, etc.), since passwords—even though they may be chosen randomly—are typically much too short for these kinds of tests to be applicable. As a consequence, the algorithm tends to *under*rate passwords (consisting of random characters) and to *over*rate passphrases (consisting of random words from a word list). As mentioned above, two leading asterisk symbols (**) indicate that the entered password matches a common password.

If a password database is currently opened in PwTech, select **Add to Database** from the context menu to add a new entry with the given password.

Clicking on the **symbol** ••• hides the password characters behind symbols.

You can change the font which is used to display the password by right-clicking in the box and selecting the entry **Change Font** from the context menu.

Random Pool

The status bar on the bottom of the window displays information about the current state of the random pool, i.e., the amount of *entropy bits* (E) the pool can currently provide (e.g., "E |

396+”). Whenever you generate one or more passwords, *N·p* bits will be “consumed” from the pool (*N* = number of passwords, *p* = bits of entropy in each password).

You can provide entropy by moving your mouse, by clicking with your mouse and by typing on your keyboard. Furthermore, the program regularly collects entropy from various volatile system parameters.

The random pool can yield a maximum Shannon entropy of 256 bits. However, PwTech also shows the non-limited entropy amount, which may exceed 256 bits and may be of interest if you don’t trust PwTech’s estimations regarding the entropy content of the different sources. So the label “396+” means that you have provided 396 entropy bits so far, and that the random pool has reached its maximum entropy *according to PwTech’s estimations,* indicated by the “+” symbol. The non-limited entropy counter is increased (up to 10,000 maximum) as long as you don’t generate any passwords. Whenever you “consume” entropy from the pool, the entropy counter will assume a value less than 256 bits. Additionally, there is a *total entropy counter*, accessible in the *Help* menu, which counts the amount of entropy gathered by the program during its runtime (up to 1,000,000,000 maximum), and which is *not* decreased after consuming entropy from the pool.

You can disable the entropy information in the status bar via the main menu, *Options* | *Hide Entropy Progress*.

Main Menu



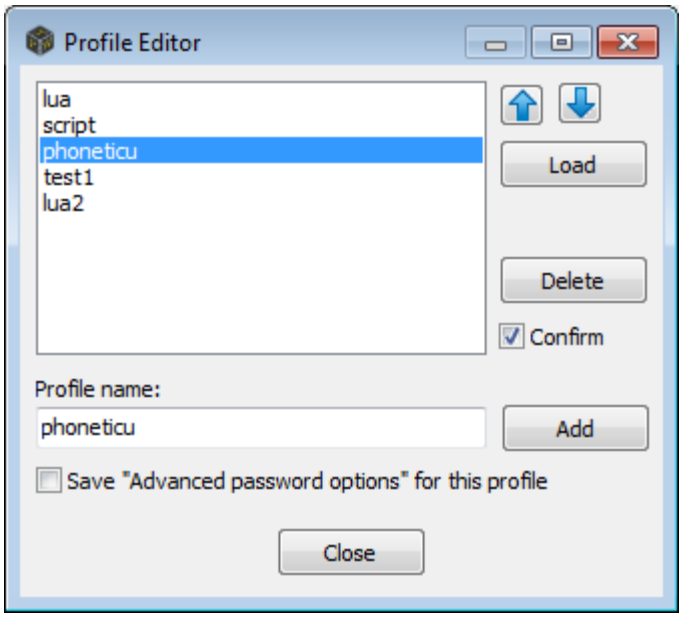
Note that some of the following functions can also be accessed via the toolbar buttons right below the main menu.

File

Profile

Shows a submenu containing all password generation profiles. To load a profile, select the respective menu item or press the designated key combination (<Shift>+<Alt>+<0>, +<1>, …, +<9>, +<A>, …, +<Z>). All the settings for generating passwords (including advanced password options, if saved additionally) will be restored according to the selected profile.

Profile | Profile Editor (<F10>)



In this dialog, you can manage your profiles, i.e., add/overwrite, load, and delete profiles. Note that the number of profiles is limited to 50, and that profiles which contain additional advanced password options are marked with "[+]".

Add/overwrite a profile: Enter the name of the profile into the **Profile name** box and press **Add**. Activate **Save "advanced password options" for this profile** if you want to explicitly store the advanced options with this profile; if you leave this option deactivated instead, the current advanced options will never be modified when loading this profile. If a profile with an identical name already exists and **Confirm** is activated, you will be asked to confirm the over-writing.

Load a profile: Select the profile of your choice and press **Load**, or double-click on the entry.

Delete profiles: Select the profile(s) you want to delete and press **Delete**. If **Confirm** is activated, you will be asked to confirm the deletion.

Move a profile: To change the position of a profile within the list, click on the **arrow buttons.**

Note that there are no *OK* and *Cancel* buttons in this dialog; all changes to the profiles are made active instantaneously, and there is no possibility to undo an accidental deletion during a PwTech session. To restore the profile settings to the "start-up state" (initial settings loaded from the configuration file on start-up), close the window, make sure that the *Options* | *Save Settings on Exit* menu item is deactivated, and restart the program.

Exit (<Alt>+<Q>)

Closes the application.

Tools

Clear Clipboard (<F2>)

Clears the text contents of the clipboard.

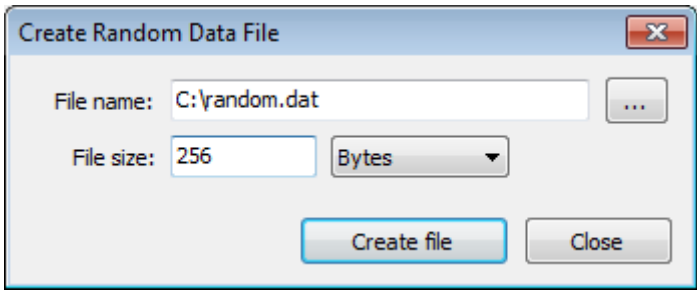
Encrypt/Decrypt Clipboard (<F3>/<F4>)

Encrypts or decrypts the clipboard text and stores the result in the clipboard. At first you have to enter a password which can be of any length. The clipboard text (if there's any) will then be en-/decrypted using a secure 256-bit key derived from this password. The encryption algorithm is AES with a key length of 256 bits. Note that the text is compressed before encryption. (For more information about the exact encryption procedure, see: *Text Encryption*.) The ciphertext will be converted to the base64 format consisting of letters (A-Z, a-z), numbers (0-9) and the symbols "+", "/" and possibly "=". In the resulting text, lines are automatically wrapped after 76 characters. It is, of course, strongly recommended to check the encryption before discarding the plaintext. If the decryption fails, you probably entered a wrong password. Another possibility is that the text is corrupted, i.e., it has been modified in some (bad) way. Note that decryption will always fail if one or more characters in the ciphertext have been altered (except for newline characters). Inserting or removing line breaks in the ciphertext is safe.

The clipboard encryption is not intended for encrypting very long texts. (Please use a file encryption utility for this purpose.) To avoid memory problems, PwTech currently limits the text length to 128 megabytes.

Important note: The encryption scheme has been changed in PWGen 2.3.0 to provide Unicode support. The plaintext is converted to UTF-8 before compression and encryption. For backwards compatibility, i.e., to decrypt ciphertexts generated with older versions, you can select the option *PWGen <2.3.0* in the password dialog when decrypting texts. It is strongly recommended to re-encrypt any existing ciphertexts with the new version.

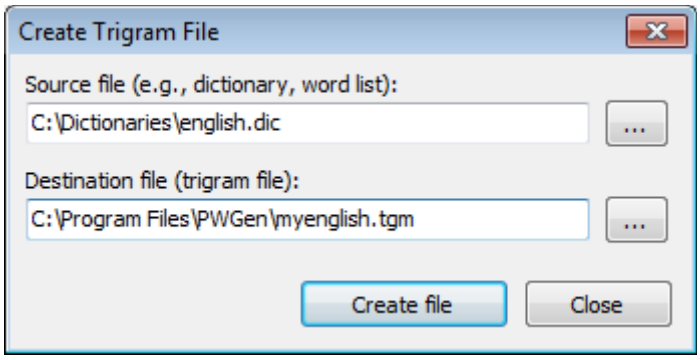
Create Random Data File (<F5>)



Creates a file consisting of purely random (i.e., *cryptographically random*) data. First, select an existing file (by clicking on the **Browse** "**...**" button) or enter the **name of the file** where the random data is to be stored. Then enter the desired **file size** in *bytes*, *kilobytes* (1,024 bytes), *megabytes* (1,048,576 bytes), or *gigabytes* (1,073,741,824 bytes), and select the appropriate list entry. Note that you can enter floating-point numbers (e.g., "0.5", "3.14", "2.25e3", etc.), which will be converted to an integer number of bytes. In case of a file size of more than 1 megabyte, you have the possibility to stop the creation process by clicking on the *Cancel* button in the progress window.

Note: The file size may now exceed the previous limit of 2 gigabytes. The new upper limit is 10 terabytes (10,240 gigabytes).

Create Trigram File (<F6>)

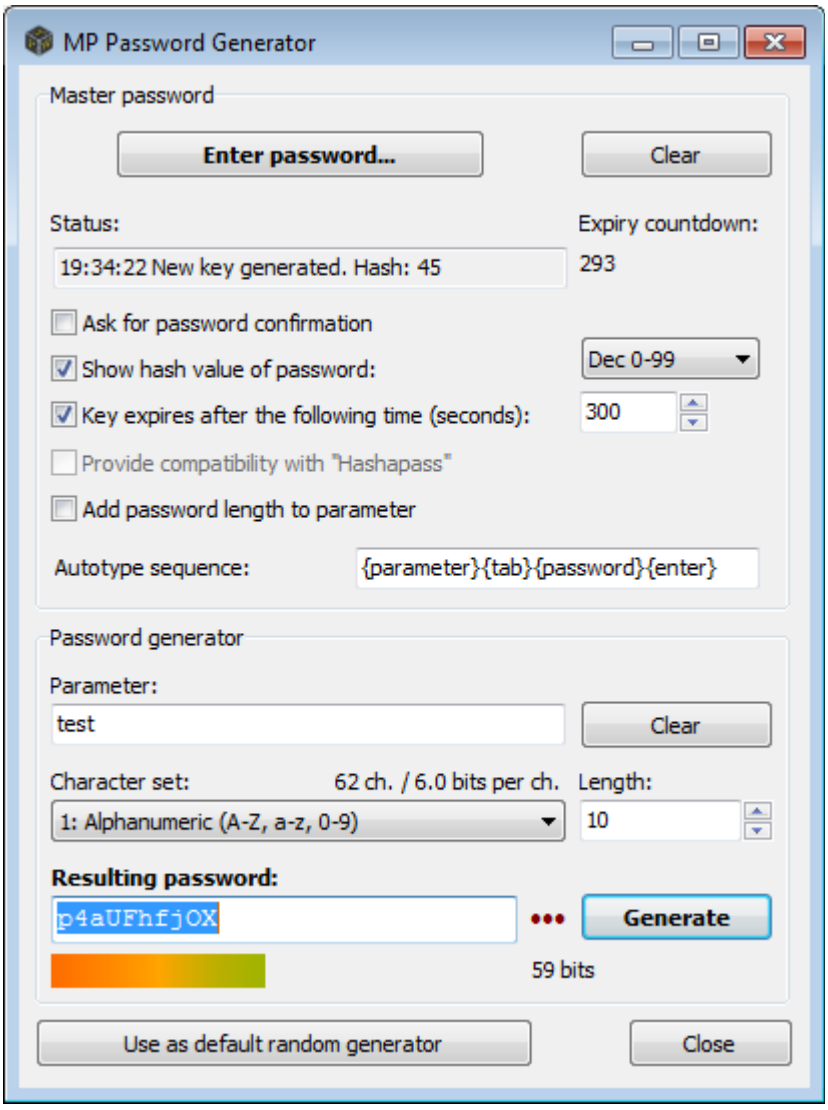


Here you can create a "trigram file", i.e., a file containing the frequencies of all possible 26^3 = 17,576 trigram (3-letter) combinations (*aaa*, *aab*, …, *zzz*). The **Source file** may be any (Unicode or ANSI) text file with words consisting of letters (lower-/upper-case) of the English alphabet (A–Z, a–z); all other symbols—even derived letters such as *ä, è, î*, etc.—are ignored! When clicking on **Create file**, PwTech will analyze the entire source file and write the trigram statistics to the **Destination file**, which should have an extension of *.tgm*. This file may be loaded via the *Advanced Password Options* dialog to generate custom phonetic passwords. If the destination file could be created successfully, PwTech shows some information about the number of trigrams evaluated and the amount of entropy per letter. If all trigrams had the same occurrence, the entropy per letter would be $\log_2 26 \approx 4.7$. However, as every language has its own "phonetic rules", the trigram frequencies tend to exhibit large variations, which lead to a decrease in the amount of entropy.

Word lists, and, even better, dictionaries are especially useful for the trigram analysis. Novels or other long texts are suitable, too, but the resulting entropy may be lower compared to dictionaries: In texts, the different frequencies of words must be taken into account, which increases the frequencies of certain trigrams and thus *decreases* the total entropy even further. (The resulting passwords might be better pronounceable, though). Typical entropy values are ~3.5 for dictionaries and ~3.0 for long texts (tested for English and German).

Example: In order to generate phonetic passwords in a specific language with Latin-based letters, it would be constructive to analyze a large dictionary (>1 megabyte) of that language. Since all non-Latin characters are ignored by the algorithm, it may be helpful to convert all umlauts, accented letters, etc. to allowed letters (a–z).

MP Password Generator (<F8>)



"MP Password Generator" (or even shorter: MPPG) stands for "Master-Password based Password Generator". This tool allows you to create (practically) unique passwords for various applications, such as websites, bank accounts, etc. Basically, it works as follows: The user provides a *master password* and a *parameter* (e.g., the name of the website). Then, PwTech uses strong cryptographic algorithms to derive a password of variable length from these user-supplied data: HMAC-SHA-256 for deriving a 256-bit key, and AES-256 in counter mode as a cryptographically-strong pseudo-random number generator (CSPRNG) for generating passwords. Thus, the MPPG can serve as a fully portable "password safe", functioning without any encrypted file storage (which is usually essential for accessing one's passwords). Instead, each password can be directly accessed by its unique identifier (= parameter). By default, PwTech uses its own method to generate passwords; however, an option is provided which enables full compatibility with the online password generator tool *Hashapass*. Let's have a look at the details of the usage:

Set master password: Clicking on the button **Enter password** opens a dialog where you can enter your master password. If the option **Ask for password confirmation** is activated, you have to enter the password twice to avoid using a wrong password due to typing errors.

Clear password: Press the **Clear** button to clear the key cache, the parameter, and the last generated password.

Once the master password has been set, a new message in the **Status** field is displayed. A time stamp is added to inform the user that the password has been updated successfully. If the option **Show hash value of password** is activated, the status message additionally contains a hash value of the password in a format that can be selected using the **drop-down list** on the right: As a decimal number in the range 0-99 or 0-9,999, or as a 16-bit (0000-FFFF) or 32-bit (00000000-FFFFFFFF) hexadecimal number. This hash may be helpful for quickly checking whether the password has been entered correctly. Checking a hash number may be more convenient than always having to enter the password twice. The default 16-bit hash has a complexity of 2^{16} = 65,536, so the probability that a wrongly typed password has the same hash as the correct one is about 0.0015%, which may be considered negligible if the amount of typing errors in the password is limited to a few positions. A possible downside is that you have to remember another character string along with your password. Note that it is not possible to derive any information about the master password from this hash.

Key expiry: For reasons of convenience, PwTech can store the master password in memory for easy re-use. This so-called "key cache" is encrypted with a random 128-bit key, which is itself stored in a different memory location. If the option **Key expires after the following time** is *de*activated, the key "expires" only when the user clears it manually, or when the window (or the whole application) is closed. If the option is activated, the key cache will be cleared some time after a new master password has been entered. You can enter a value (in seconds) in the range 0–32,767 in the corresponding input box:

- A special case applies to the number "0", which means that the key will not expire before a new password has been generated for the first time. In this case, only the master password will be cleared, whereas the parameter string and the generated password will remain unaffected.

- In all other cases, a countdown for the expiry of the key will be displayed in the field next to the *Status* box as soon as the user has entered a master password. When the countdown reaches 0, the key cache, the parameter and the lastly generated password will be cleared.

Hashapass compatibility: Some users may want to be independent of PwTech with respect to the MPPG; for example, when traveling, or when using other peoples' computers, it may not be possible to install or even execute PwTech. For these users, PwTech offers the option **Provide compatibility with Hashapass**: The functionality is comparable to PwTech's MPPG (although I think that the latter is more secure—*see table below for a detailed comparison*), but Hashapass has the advantage that its website is accessible via the world wide web and should be compatible with all popular browsers, independent of the operating system (Windows/Unix/Mac OS/...). Hashapass generates passwords in offline mode "*in the safety of your web browser, without any information being transmitted over the Internet*" (cited from the website). Unfortunately, the password length is limited (fixed) to 8 characters (= 48 bits of security), and key caching is not supported, so the master password always has to be re-entered by the user to generate a new password. Another downside is that the master password has to be stored in memory (albeit in encrypted form), whereas with PwTech's own method, a 256-bit hash value

of the master password is cached, which does not reveal any information about the master password itself.

Note: Due to the differences in key caching the selection of the *Hashapass* option cannot be changed anymore as soon as the master password has been set. To change the selection, clear the key cache (via the *Clear* button), (de)select the option and enter the password again.

To ensure that all characters of the password depend on the preferred password length, activate the option **Add password length to parameter** (not available in Hashapass mode). Then the length value is appended to the parameter string, so that the generated random number sequence is different for different length values. If the option is deactivated, the random number sequence is identical for all length values, and will simply be truncated or elongated as the length is decreased or increased, respectively.

Enter a unique **Parameter** into the corresponding input box. It can be any kind of character string, for example, the name of a website or application, a URL, etc. It is best to choose a distinct, yet easy to memorize parameter. Note that the box can act both as a drop target (receive drag & drop text from other applications) and as a drop source by clicking on the *Parameter* label and dragging the text into other applications.

Choose a **Character set** from the drop-down list. Currently, PwTech provides six different sets, of which some are available without ambiguous characters (`B8G6I1l|0OQDS5Z2`). You may wonder why there isn't any option to define custom character sets. The reason is that the MPPG functionality should be as portable as possible, that is, it should work without having to remember a plethora of settings such as the exact definition of the character set. Even if you don't carry a personalized version of PwTech with you all the time, you can download a new copy from the Internet and execute it on Windows (or via *Wine;* installation is not necessary!). The only settings you have to remember are the type/index of the character set and the password length. If you don't know which character set and/or which password length to choose, simply use the default settings.

Enter the desired **Password length** (number of characters) into the corresponding input box. For security reasons, I recommend using the default value of 16 characters (or more; at least 12). Unfortunately, some applications limit the password length to only 8 characters, which may be too short for sensitive data or applications (e.g., online banking accounts). Simply use the first 8 characters of the generated password in such cases.

Click on the **Generate** button to generate the password which results from the unique combination of the master password and the parameter string. The quality of the password (measured in bits) is displayed as a number and illustrated by the length of the "password quality bar" (see: *Generate Single Passwords* for more details) below the password box. Note: The font for displaying the password is the same as that for the password box in the main window.

Drag & Drop: Passwords can be dragged & dropped just like those generated in the main window. Move the mouse cursor to the security bar or to the "*Resulting password*" label (on top of the password box) and hold the left mouse button to initiate a drag & drop operation. Conversely, you may also drop text from other applications into the box.

Autotype: Right-click in the password box to open the context menu and select **Perform Autotype** to let the program automatically type the password into another application (see: *Configuration* | *General* for more details on the autotype feature).

Store in database: Right-click on the password box and select **Add to Database** to store the parameter—password pair in the currently opened PassCube database. The menu item is only enabled when a database is opened.

Click on the **symbol** ••• to hide the password behind symbols.

The button **Use as default random generator** enables you to make PwTech's random generator completely "deterministic" by replacing the default generator (random pool) with the pseudo-random number generator (PRNG) which is used for the MPPG. In this case, all passwords and random data files generated with PwTech are "deterministic" in a sense that they can be reproduced *exactly* by using the same master password and parameter again. (Note that the output of this PRNG is still cryptographically-secure, since it is based on AES with a 256-bit key.) In contrast, the random pool was designed to never generate the same output twice (at least the probability of a repeat should be so low that it is practically impossible)[1]. By using this option you can extend the "password hasher" functionality *practically to the entire application*. Note that the random pool will still be active and gather entropy from user inputs and system-specific parameters, but it will not be used for generating passwords and random data files.

For technical reasons, this button is only accessible if the *Hashapass compatibility* option is *not* activated. Otherwise, you have to clear the key, deactivate the option and re-enter the password to gain access to it.

Please read the following notes carefully:

- I cannot guarantee that the algorithms for generating passwords in the main window ((phonetic) passwords, passphrases, formatted passwords) will never change again in future program versions. So it *might* be possible that some kinds of passwords generated in the main window using the MPPG option cannot be reproduced in some future version of PwTech. However, this guarantee is given for the MP Password Generator itself; if the MPPG algorithm has indeed to be changed due to a (severe) bug in the code, I will at least provide an option for backwards compatibility.

- Never forget to deactivate the deterministic random generator as soon as you don't need it anymore (a warning message is displayed in the window caption to remind you of that). It can be deactivated via the menu item *Tools* | *Deterministic Random Generator* | *Deactivate*, or by pressing F7. Upon deactivation, the random pool will be re-used as default random generator.

1 From a theoretical standpoint, the random pool is also a "pseudo-random" number generator because it uses deterministic algorithms to generate random numbers. However, the entropy provided by the user and the computer system (especially the high-resolution timer provided by the CPU) make the construction more and more "indeterministic".

Reset: The deterministic random generator can be reset (i.e., set to its initial state right after activation) via the menu item *Tools* | *Deterministic Random Generator* | *Reset*, or by pressing <Ctrl>+<F8>.

The button **Close** first clears the key cache and all sensitive input fields (parameter and password box) before closing the window. Note that the deterministic random generator may still be active and has to be turned off separately (simply press <F7>).

CAUTION: I strongly recommend using a "physical" password safe such as PwTech's own *PassCube*, *KeePass* or *Password Safe* in addition to the MPPG. This double-tracked approach protects against memory loss regarding parameters (though not the master password, of course), and ensures that a complete list of all your parameter–password combinations (maybe with additional information like user names and personal comments) is available as a backup. If you don't want to use such a password safe, you can also keep a list of all parameters in a simple (encrypted?) text file.

The following table lists the differences between PwTech's MP Password Generator and the Hashapass password generator:

Parameter	PwTech's MPPG	Hashapass
Password length	1–32,767 characters	8 characters (fixed)
Password character set	Currently 6 sets to choose from	Base64 (*A-Z, a-z, 0-9, +, /, =*)
Password security	Up to 256 bits (variable)	48 bits (fixed)
Mode of password generation	1) Derive a 256-bit hash *H* from the master password by applying HMAC-SHA-256 (to avoid caching the password as plaintext); 2) Derive a 256-bit key *K* from *H* and a user-defined parameter *P* by applying PBKDF2 (based on HMAC-SHA-256) with 8192 iterations; 3) Initialize AES with *K* and use AES in counter mode to generate random numbers; change key after every 1 megabyte of random output	1) Apply HMAC-SHA-1 to the master password and the parameter; 2) Take the first 48 bits and convert it to base64, which yields the final 8-character password
Cryptographic primitives used	1) SHA-256 (*256-bit one-way hash function*) 2) AES (*128-bit block cipher with 256-bit key*)	SHA-1 (*160-bit one-way hash function*)
Information stored in encrypted key cache	256-bit hash *H* of the master password	Master password
Unicode support	Full support	Partial support (only the lowest significant byte of each Unicode character is used, so some non-ASCII characters are not distinguishable when used together in a string)

Availability	PwTech (32-bit & 64-bit), on other operating systems possibly via *Wine*	Website, should work with all popular browsers; PwTech and possibly other applications

Deterministic Random Generator

This menu controls the so-called "deterministic" random generator, which can be activated via this menu or in the *MP Password Generator* dialog via the button *Use as default random generator*.

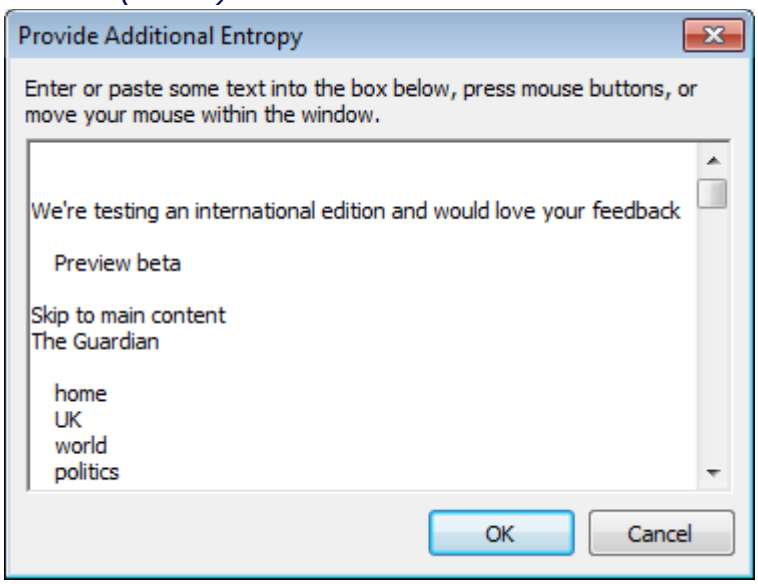
Set up (<Ctrl>+<F8>): Initializes the deterministic random generation with a user-supplied password and sets the resulting random generator as the default one for the entire application. This setup is equivalent to the setup via the MP Password Generator with the same password and an empty parameter.

Reset (<Ctrl>+<F7>): Resets the random generator to its initial state, thus allowing you to reproduce the entire sequence of random numbers generated so far.

Deactivate (<F7>): Destroys this random generator and resets the random pool as the default random generator.

Provide Additional Entropy

As Text (<F9>)



Opens a window with a text box where you can enter some text (by pressing keys, pasting text from the clipboard, or via drag & drop from other applications), press mouse buttons, and move your mouse in order to feed the random pool with entropy (i.e., data which is indeterministic to a certain extent). When you're finished, press **OK** to add the text to the random pool and evaluate it with respect to its entropy content: The text is first converted to UTF-8 encoding and compressed; the compressed data is then fed into the random pool, and the length of the compressed data block in bits, downgraded by a safety factor, is added to the entropy counter of the random pool. Clicking on **Cancel** skips this procedure.

Note that each interaction with your keyboard or mouse—no matter in which window—is added to the pool instantaneously, indicated by discrete increments in the "Entropy bits" counter (displayed in the main window). For these kinds of events, the entropy is mainly determined by the quality/precision of their time stamps. The text in the text box as a whole may contain some additional entropy, provided that the content is "indeterministic" to a certain degree. To fulfill the last requirement, it is recommended to paste longer texts from volatile, non-static sources (e.g., log files, chat protocols, frequently updated web pages, …) into the box. The longer the text, the more precise the entropy estimation will be.

From File (<Ctrl>+<F9>)

Select a file the contents of which you want to incorporate into the random pool. The file is read in 64-KB chunks, which are compressed and then added to the pool (similar to the procedure described above). PwTech assesses each compressed byte with 4 bits of entropy (corresponding to a safety factor of 0.5).

Options

Configuration (<F11>) 

Allows you to access the main configuration of the program. In the dialog, you can select between various pages: **General**, **Security**, **Hot Keys**, **Files**, **Updates**, **Language**, and **Database**.

General

Change font for the GUI controls: Changes the font which is used to display the text in most of the GUI controls in the application. The font of the password box in the main window and of the password list window cannot be changed via this function; please use the corresponding functions in the popup menus of these controls for this purpose. The default font is Tahoma with a size of 8 pt.

Show system tray icon constantly: Shows PwTech's program symbol in the system tray (next to the system clock). Clicking on the symbol will open a menu where various functions of the program are accessible.

Minimize program to system tray: If activated, PwTech's taskbar button will be hidden when the program is minimized, and the program symbol will be displayed in the system tray instead. If *Show system tray icon constantly* is deactivated, the symbol will be removed when the application is restored.

Minimize before performing autotype: The so-called *autotype* feature sends any character string such as passwords and user name—password combinations to other applications, which may be useful for automatically entering login/authentication data into website forms, credential dialogs, etc., without having to apply multiple copy—paste operations. If the option is activated, PwTech is minimized when an autotype operation is requested, and the selected character sequence is immediately sent to the application that gets focus after PwTech has been minimized. If the option is deactivated, a background thread is started, which waits until another window (within PwTech) or application is focused by the user before finally sending the auto-

type sequence. Note that this thread times out after 10 sec, thus you have 10 sec to select the target window or application before the thread destroys itself.

Format of autotype sequences: Apart from sending fixed character sequences such as passwords, the sequence to be sent can be formatted in a more complex way by using placeholders for fields (such as *title*, *user name*, *password*, etc.) and virtual keys (such as <ENTER>, <TAB>, etc.). Such placeholders have the format {*placeholder*}, whereas verbatim keys can be written as-is in the sequence. For example, {username}{tab}{password}{enter} first sends the characters of the field *user name*, followed by a <TAB> key, then sends the characters of the field *password*, followed by an <ENTER> key. The following placeholds are supported:

Placeholder (case-insensitive)	Description
title	Database field: *Title*
username parameter	Database field: *User name* equivalent to MP Password Generator field: *Parameter*
password	Database field: *Password*
url	Database field: *URL*
keyword	Database field: *Keyword*
keyvaluelist	Database field: *Key-value list* (formatted as key1=value1, key2=value2, …)
notes	Database field: *Notes*
tags	Database field: *Tags* (formatted as tag1,tag2,…)
tab	Key <TAB>
return enter	Key <ENTER>
ctrl	Key <CTRL>
backspace	Key <BACKSPACE>
clear	Key <CLEAR>
shift	Key <SHIFT>
alt	Key <ALT>
pause	Key <PAUSE>
capslock	Key <CAPSLOCK>
escape	Key <ESCAPE>
space	Key <SPACEBAR>
pageup	Key <PAGEUP>
pagedown	Key <PAGEDOWN>
end	Key <END>
home	Key <HOME>
left	Key <ARROWLEFT>
right	Key <ARROWRIGHT>
down	Key <ARROWDOWN>

`up`	Key <ARROWUP>
`select`	Key <SELECT>
`print`	Key <PRINTSCREEN>
`execute`	Key <EXECUTE>
`snapshot`	Key <SNAPSHOT>
`insert`	Key <INS>
`delete`	Key <DEL>
`help`	Key <HELP>

Autotype delay between characters: To ensure that all characters are processed properly by the target application, it is recommended to insert a delay of several milliseconds between the simulated keystrokes. For normal Unicode characters, a delay of 50—100 ms (default 75 ms) should be sufficient. For special keys such as <ENTER> or <TAB>, PwTech uses a minimum delay of 200 ms.

Ask before exiting application: Displays a message box asking for confirmation before closing the application.

Security

Encryption algorithm for generating random data via random pool: PwTech currently offers two completely different encryption algorithms to be used by the random pool, namely the block cipher *Advanced Encryption Standard* (AES aka Rijndael) in counter (CTR) mode, and the stream cipher *ChaCha20*. Both ciphers have a key size of 256 bits and generate random data by continuously incrementing and encrypting a counter value. Both algorithms are considered very secure and can safely be used as random generators.

- **AES-CTR:** The "Rijndael" family of block ciphers was designed by J. Daemen and V. Rijmen in 1998, and the variant with key sizes of 128, 192, and 256 bits and a block size of 128 bits was selected as the winner of the AES competition in 2000. It is now widely used as an industry standard. To generate random data, PwTech operates 256-bit AES (AES-256) in counter mode by continuously incrementing and encrypting a randomly chosen 128-bit number (the counter). Hence, the resulting stream is a pseudorandom permutation of the numbers $0..2^{128}-1$.

- **ChaCha20:** This is a stream cipher designed by D. J. Bernstein in 2008, which is closely related to the *Salsa20* cipher that was submitted to the eSTREAM project by Bernstein in 2005. It has a key size of 256 bits and an internal state of 512 bits, with the key being a part of the internal state. A random stream is generated by performing cycles of (a) encrypting the internal state, (b) using the result as random 512-bit output, and then (c) incrementing an internal 64-bit counter value, which is also part of the internal state. Thus, like AES-CTR, ChaCha20 functions as a pseudorandom permutation, but with a considerably larger complexity (2^{512}) compared to AES-CTR.

Due to the larger internal state of ChaCha20 (512 bits) compared to AES-CTR (128-bit block size), PwTech uses ChaCha20 as the default encryption algorithm for the random pool. It is also faster than AES in software implementations.

Note that for text encryption as well as for the deterministic random generator, PwTech still uses AES-256 exclusively. Support for ChaCha20 may be added in future versions of the program. As of version 3.3.0, databases may also be encrypted with ChaCha20.

Test password against list of common passwords: If activated, PwTech loads the list of common (bad) passwords contained in the file *common_passwords.txt* on startup and checks every generated password against this list. Matching passwords are marked with two asterisks (**) in the label below the password box in the main window. PwTech comes with a list of the 10,000 most common passwords[2], but in principle any list (with at least two entries, separated by newline characters) can be supplied in the file *common_passwords.txt.*

Clear clipboard after the following time (seconds): Sensitive data copied to the clipboard is normally only used for a short amount of time (several seconds) and may pose a risk when residing in the clipboard memory beyond its designated use (i.e., pasting). Activate this option to automatically clear the clipboard contents after a certain number of seconds (to be defined in the input box below the checkbox) whenever text from password boxes is copied to the clipboard.

Clear password box after the following time (seconds): Automatically clears the text in the password box in the main window after a certain amount of seconds (to be defined in the input box below the checkbox). The countdown starts when generating a new password by clicking on the "Generate" button in the main window.

Hot Keys

The "hot key" feature allows you to associate special key combinations (shortcuts) with various functionalities provided by PwTech. Whenever you press the designated key combination while PwTech is running (in the background), PwTech is notified and performs an action of your choice: The main window can be shown or restored (if minimized), passwords can be generated, the password manager can be opened, etc.

First, select an **action** to be associated with a hot key:

Activate **Show/restore main window** to show or restore (if minimized) PwTech's main window when the hot key is pressed. Additionally or alternatively, you can select an action from the drop-down list:

Action	Remark
No further action	Only valid if *Show/restore main window* is activated.
Generate single password in main window	Equivalent to pressing *Generate* button.
Generate multiple passwords	
Generate password and copy it to the clipboard	*Self-explanatory*
Generate password and show it in a message box	Password is shown in an interactive message box that allows you to copy it to the clipboard

2 Taken from D. Miessler's repository at https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials. You can find a multitude of password lists there.

	(by pressing *Yes*) or generate a new password (by pressing *No*). Note that the main window will always be restored, so that the message box can be placed in the foreground. When you close the box afterwards, the program will be minimized again, if necessary.
Generate password and perform autotype	Password is immediately autotyped, provided that the target application is not PwTech (you have 10 sec to activate the target window).
Show MP password generator	Show or restore window.
Show password manager	
Search database for keyword using window title	If database is open, get title of currently active window and search database for the first entry the keyword of which matches the title (search is case-insensitive), then show the found entry in the password manager window.
Search database for keyword and perform autotype	Like previous action, but instead of showing the found entry in the password manager window, perform autotype for this entry.

Second, press the desired **key combination** in the **Shortcut to assign** box, for example <Ctrl>+<Alt>+<P>, <Ctrl>+<F12>, <Alt>+<F8>, …, and press **Add** to add the hot key to the list. To remove hot keys, select the entry or multiple entries in the list and press **Remove**.

The hot keys will be registered and become active when you close the configuration dialog via the *OK* button.

Files

Character encoding for text files: Unicode characters are encoded as UTF-16 internally in Windows, but when writing Unicode text to a file, it may be necessary to transcode the characters in order to ensure compatibility with other applications/systems. PwTech supports ANSI (non-Unicode!), UTF-16 (Little Endian), UTF-16 Big Endian, and UTF-8 (see *Unicode Support* for more details on Unicode and the different types of encodings).

Note that in some Windows applications such as *Notepad* (which also supports UTF-16 and UTF-8 encoding of text files), the UTF-16 encoding is referred to as "Unicode", which is misleading because UTF-8 is as "Unicode" as UTF-16; both are valid Unicode character encodings!

Newline character sequence: To indicate the end of a line and start of a new line, Windows uses the character sequence "carriage return" followed by "line feed" (ASCII characters 0d and 0a in hexadecimal notation, escape sequence "\r\n"), whereas Unix and Unix-like systems simply use a "line feed" character ("\n").

Updates

Automatic check for updates: Enables automatic update checking at regular intervals, i.e., *daily*, *weekly*, or *monthly*. Checks are performed on start-up, and a message will be displayed if a new version is available. You can also *disable* this option and check for updates yourself (see: *Help* | *Check for Updates*).

Language

Select language: Changes the program language. You will receive a warning message if the language version is not compatible with the program version. The program has to be restarted to load the language file. If you downloaded a language file from the PwTech homepage, extract it into the program folder. PwTech will find it on start-up and add the language name along with the version number as an item to the drop-down list.

Database

This page provides various options and parameters to customize the password database/manager functionality.

Clear clipboard on minimize/exit: Clears the clipboard when the password manager window is minimized/closed.

Lock database when minimizing application or database window: Locks the database when the database window or PwTech's main window is minimized.

Lock database after the following idle time (seconds): Automatically locks (closes) the database if the database GUI has been idle (i.e., has not received any productive user input) for a given amount of seconds, which is to be entered in the input box below the checkbox. Activate **Save automatically** to automatically save pending changes before the database is closed. If this option is *disabled*, you will have to confirm the save operation (or reject it) through a message box.

Create backup of the database before saving: Creates a backup of the current version of the database file before writing the changes to the file. The naming of the backup files can be based on consecutive numbering or timestamps:

- If **Number backups consecutively** is selected, the name of the backup file will include a number (1-999) that is incremented each time a new backup is created. The maximum number of backups can be changed in the input box next to the checkbox. The format of the backup file name is {*originalfilename*}_*xxx*.bak, with {*originalfilename*} being the file name of the database without file name extension, and *xxx* being a 3-digit number (e.g., database: johndoe.pwdb, backups: johndoe_001.bak, johndoe_002.bak, …). If the maximum number of backups has not been exhausted yet, PwTech will select the lowest unused number for the file name. Otherwise, it will overwrite the oldest backup file.

- If numbering is disabled, the name of the backup file will include a timestamp, resulting in the file name format {*originalfilename*}_*YYYYMMDDhhmmss*[_*nnn*].bak, with *Y* = year, *M* = month, *D* = day of the month, *h* = hour, *m* = minute, *s* = second, and *n* = millisecond (optional, only if a backup file with a name corresponding to the regular format already exists). In contrast to the previous option, the number of backups cannot be limited.

Open window on startup: Opens the password manager window when PwTech is started.

Open last used database on startup: Opens the last used database when the password manager window is opened.

Save automatically: Immediately saves the database after change operations:

- **After adding/modifying an entry:** Whenever an entry has been modified or a new entry has been added to the database.
- **After every change:** *Self-explanatory*.

Default autotype sequence: The default sequence to be used when performing the autotype function for database entries that lack a customized autotype sequence. Use placeholders to insert title, user name, password, etc. See table above (*General* page) for a complete list of supported placeholders, including special keys.

Always on Top

Select this option to make PwTech the top-most window on the screen, meaning that the main window and subordinate windows will stay on top even when the application is deactivated and another application gets focus.

Known issues: When this option is actively *disabled* during runtime, the "Quick help" window, which is opened by clicking on one of the corresponding buttons in the main window, may be empty. Please press the button again to display the help text correctly.

Save Settings on Exit

If checked, all program settings will be saved in an .ini file (default: *PwTech.ini*) when the user exits the program.

Hide Entropy Progress

By default, PwTech displays the amount of entropy collected so far (consisting of user inputs and volatile system parameters) in the status bar of the main window, in the lower right corner (see: *Random Pool*). Enable this option to hide this information.

Save Settings Now

Saves the settings instantaneously.

Help

Open Manual (<F1>)



Opens this manual.

Visit Website

Opens the homepage of PwTech in your default web browser.

Get Translations

Opens the website providing access to language files for PwTech in your default web browser.

Donate

If you like PwTech and use it on a regular basis, please consider making a donation. Donors will receive a *donor key* (see below), depending on the amount of the donation.

Enter Donor Key

Use this dialog to enter the donor key that you received after making a donation to the project. This donor key will change the edition of PwTech from *Community* to *Donor* and deactivate the message box asking for a donation, which is shown with a certain frequency at the start of the program.

Check for Updates

Connects to the Internet to check if a new program version is available. If so, you may be directed to the download page of this version.

Timer Info

Shows information about the high-resolution timer (HRT) called by PwTech whenever you press a key, move your mouse, or click with your mouse (refer to *High-Resolution Timer* for more in-depth information). "*N/A*" means that a HRT is not available on your system; a low-resolution system date and time value is used instead, providing a resolution in the millisecond range (roughly 1-15 ms). Note that it is recommended to run PwTech on systems where a high-resolution timer (*RDTSC* or *QueryPerformanceCounter*) is available. The current 64-bit value (ranging from 0 to 18,446,744,073,709,551,615) of the timer is also displayed in the message box.

About

Shows copyright information about the program.

Additional Menus

System Tray Menu

The system tray menu, which opens when you click on the PwTech symbol in the system tray, allows you to access some useful functions of the program. In addition to the self-explanatory functions and those already explained above, there are two new menu items that are only available in this menu:

Generate Password

This function quickly generates a password using the settings given in PwTech's main window and copies it to the clipboard. It's like clicking on the *Generate* button with the difference that the password is not displayed but copied to the clipboard instead. This quick generation may be useful whenever you quickly need a password.

Generate and Show Password

Generates a password and displays it in a message box before copying it to the clipboard. In the message box, choose *Yes* to copy the password to the clipboard, *No* to generate and dis-

play a new password, or *Cancel* to cancel the process. Note that the password won't be shown in the message box if it is very long (>1000 characters).

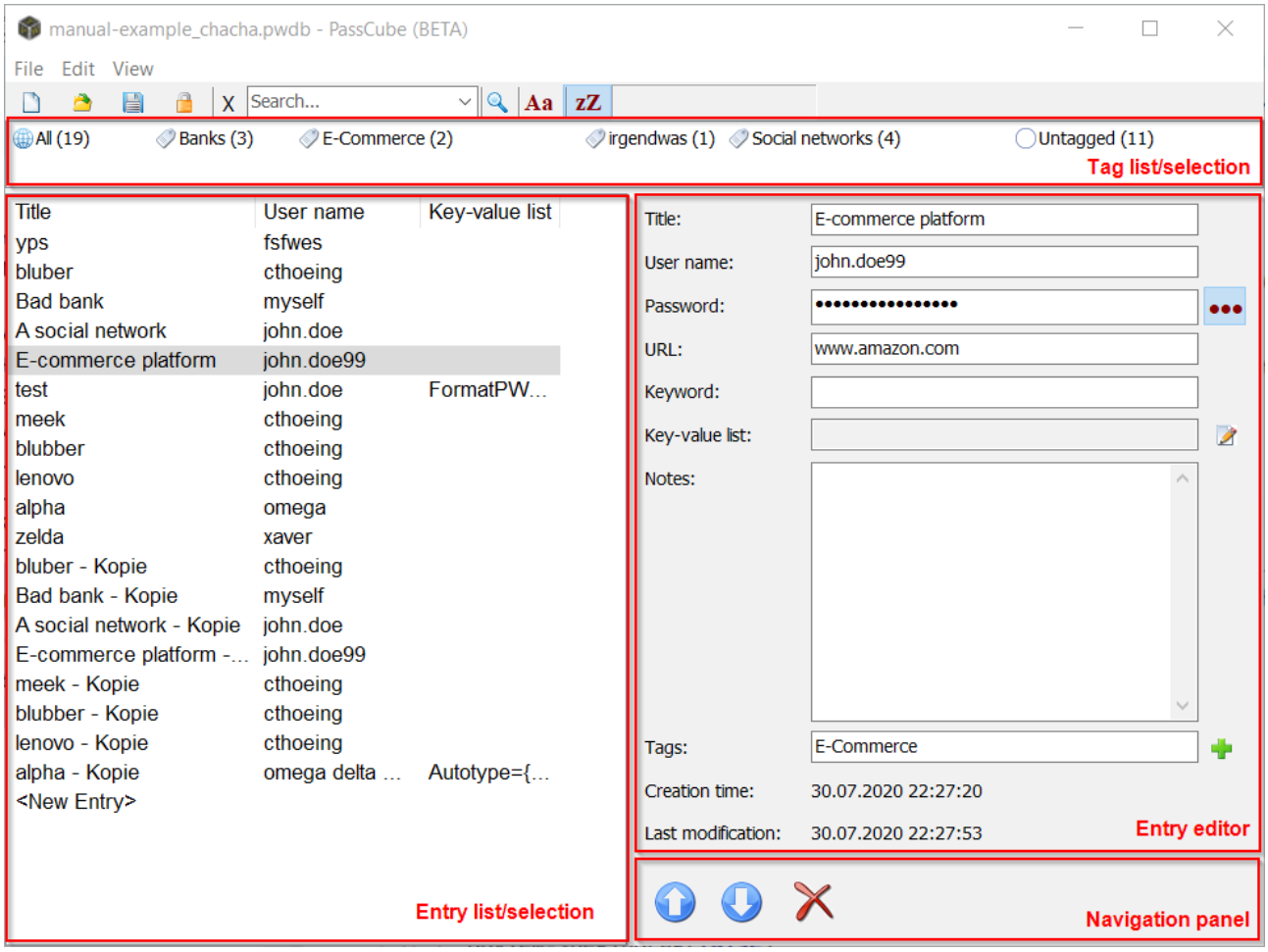
Generate and Autotype Password

Generates a password and automatically types it into the target application (for more details on the autotype feature, see: *Configuration* | *General*).

PassCube Password Manager

As of version 3.0.0, PwTech provides a password manager/password safe functionality named *PassCube*, which allows storing passwords—along with additional information such as user name, website, notes, etc.—in encrypted databases. These databases are fully encrypted using the AES encryption algorithm with a 256-bit key.

Note: *Although PassCube has been tested throroughly by the author, it is intentionally attributed a Beta state to indicate that further extensive testing is needed for optimal performance. Please help improve this feature by testing and providing feedback. The Beta state will be removed in a future version.*



The user interface of the PassCube manager has been designed for easily viewing, editing, and adding entries, all within a single window. For this purpose, the window area is separated into four panels:

- A **list of the available tags** on top of the window. Multiple tags can be selected to filter entries that match the selected tags and show them in the *list of entries* below.
- A **list of database entries** on the left side. When a single entry is selected, its details can be viewed and edited in the right *entry editor*. When multiple entries are selected, they can be deleted or rearranged within the list.

- An **entry viewer and editor** on the right side. As long as the selected database entry remains unchanged, the *navigation panel* below can be used to select the previous or next entry in the entry list or delete the current entry. When one of the available fields is changed, the navigation panel is replaced with *Accept* and *Cancel* buttons for accepting or declining the changes made to the entry.

- A **navigation panel**, which allows going forwards and backwards within the entry list on the left side. In case there are any pending changes, the panel features *Accept* and *Cancel* buttons instead of arrow (navigation) buttons, and most parts of the user interface are blocked until the changes are either accepted or rejected.

The panels (except for the navigation panel) can be resized by moving the splitters between the panels and by changing the overall window width.

Database File Handling

Create or open

To create a new database, select *File* | *New* from the main menu or press the **button**. You will be asked to enter a master password for the database.

To open an existing database, select *File* | *Open* or press the **button**. You will be asked to select a file and enter the master password to unlock the database.

Note: If another database is already opened, it will not be closed until the password for the new database has been entered (if a new database is to be created), or until the selected database has been opened successfully (if an existing database is to be opened).

Save

Select *File* | *Save* to save the changes to the current database file, or use *File* | *Save As* to select another file. If the (selected) file already exists and the backup option is enabled (see: *Configuration* | *Database*), a backup of the file will be created first before writing the changes to the file.

Close

Select *File* | *Close* or close the PassCube window to close the database.

Lock/Unlock

Select *File* | *Lock* or press the **button** to lock the database, meaning that the database is closed normally, but its file name is kept in memory, so that it can easily be re-opened by selecting *File* | *Unlock* or by pressing the corresponding button.

For options concerning the lock feature, see *Configuration* | *Database*.

Export to other file formats

Currently, PassCube databases can be exported to the plaintext (unencrypted) **CSV format** (comma-separated values). The CSV format corresponds to a table with columns separated by a comma (delimiter) and rows separated by a newline character sequence (\n or \r\n, depend-

ing on the configuration). Every item in the table is enclosed with quotation marks, so that commas that are part of the item text will not be treated as delimiters. Also, quotation marks belonging to the item text are converted into double quotation marks. The first line (row) of the file contains the column headers. The following lines contain the actual database entries.

Click on *File* | *Export To* | *CSV file* to start the export. After selecting the file to store the data, you will be asked to select the specific fields (Title, User name, Password, …) to export. Only these fields will be written as plaintext to the file.

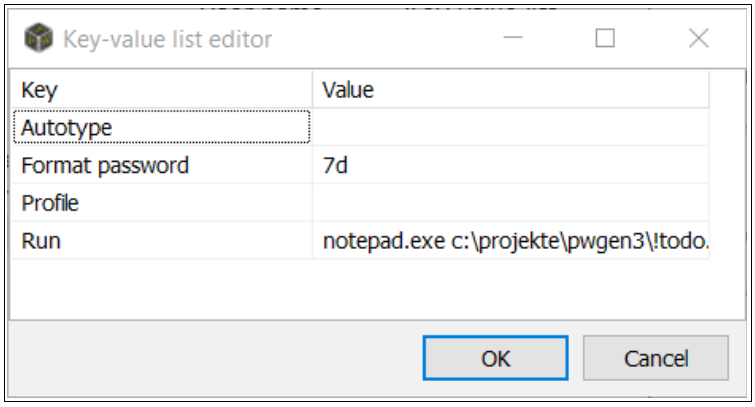
Working with the Database

Add new entry

To add a new entry, select the last entry entitled "*<New Entry>*" from the list on the left side. Then you can start filling out the fields in the edit panel on the right side. Once you have made your first change, PassCube turns into an "edit mode", and the navigations buttons are replaced with **Accept** and **Cancel** buttons. Most of the other functionalities of the database are now blocked—you have to finish editing the current entry by clicking on one of these buttons before the database becomes fully functional again (the *File* menu remains accessible.)

The available fields are listed in the following. All fields are optional, but note that you have to specify at least a *Title* or *Password* to add a valid entry!

Field	Description
Title	*Self-explanatory*
User name	
Password	
URL	Web address associated with this entry. Should be readable by a web browser.
Keyword	Keyword or expression that may be found in the title of the window of another application (such as web browser) and is characteristic of this entry. When using a PassCube-related hot key (see: *Configuration* | *Hot Keys*), the keywords of all entries are matched against the title of the currently focused window, and the first match (i.e., keyword/expression is found within the window title) is selected.
Key-value list	Provides access to further, optional fields, which are associated with certain functions of the program: • **Autotype:** Custom autotype sequence for automatically typing fields of the entry into another application. • **Run:** Custom command to be executed when the *Run* function is selected for this entry. • **Profile:** Password generation profile to be loaded in PwTech's main window when generating a new password and inserting it into the password box by *right*-clicking on the "hide password" button of the password box. • **Format password:** Format sequence for generating a password (see: *Format Password*) by right-clicking on the button next to the password box. Is only active if the "Profile" field is not specified.

	The corresponding box in the entry editor is read-only and displays the list of key-value pairs as a string with the format "*Key1=Value1, Key2=Value2, ...*". To edit the list, press the button on the right side, which will open a new window with a key-value table:  Enter the values in the second column of the table and press **OK** or **Cancel** to accept or reject the changes, respectively. Parameters with blank (empty) values will be removed from the list.
Notes	Any kind of text, may consist of multiple lines.
Tags	List of custom tags, separated by commas (,) or semicolons (;). Tags may contain any characters (including spaces) except commas and semicolons, and must not be longer than 30 characters. Tags that already exist in the database can be added to the list by clicking on the button next to the input field and selecting an item with the desired tag name from the popup menu.
Creation time	Time when the entry was created (64-bit precision).
Modification time	Time when the entry was last modified (64-bit precision).

Left-click on the **button** to toggle between hiding the password behind symbols and showing it as plaintext. By right-clicking on the button, you can generate a random password using the current settings in the main window (alternatively, specify a custom profile in the *Profile* field under *Key-value list*, see table above) and insert it into the password box.

Edit options

Options for editing entries and executing functionalities of a selected entry are available under the *Edit* menu. Some of these options are also available in the context menu of the list view (via right-click).

Copy user name/password: Copies the user name/password of the selected entry to the clipboard.

Open URL: Opens the URL of the selected entry in the default web browser.

Run: Executes the command specified in the *Run* field of the selected entry (see table above).

Perform Autotype: Automatically types certain fields of the selected entry into another application (see: *Configuration* | *General* for details about the autotype feature). If the *Autotype* field of the entry is not specified, the default sequence *User name*—<Tab>—*Password*—

<Enter> will be used (the default autotype seqeuence for database entries can be changed in the Configuration).

Add Entry: If possible, selects the last entry with the title "*<New Entry>*" in the list view (see: *Add new entry*).

Duplicate Entry: Duplicates the selected entry/entries. The suffix "- *Copy*" will be appended to the title of a duplicate.

Delete Entry: Deletes the selected entry/entries. Note that it is not possible to undo a delete operation. If you accidentally deleted an entry, save the changes to another file (via *File* | *Save As*), and re-open the original file. Alternatively, you can try opening a backup file to recover the entry.

Select All: Selects all entries that are currently shown in the list view.

Rearranging entries

By default, the database entries in the (unsorted) list view are listed in the order in which they were created, i.e., oldest entry first. You can change the position of an entry via *Edit* | *Rearrange Entry* and move the entry to the top or bottom (i.e., first or last) position, or move it up or down by one in the list. Alternatively, you can move an entry to any position using a drag & drop operation: Select the entry, hold the left mouse button, and drag the entry to the desired position, before releasing the button.

Note that rearranging is only possible when the list is unsorted.

Working with tags

All tags that have been assigned to the database entries are displayed in the list of tags on top of the window. Upon selecting a tag from the list, the associated entries are filtered in the list of entries below. When selecting multiple tags, entries matching at least one of the selected tags are filtered (combination by logical "OR"). Selecting the first field "All" shows all database entries, including untagged entries. The field "Untagged" filters entries that do not have any tags assigned (if there are any—otherwise, the field is omitted). Each tag field is displayed in the format "*name* (*N*)" with its *name* and the number of entries *N* associated with this tag.

When adding a tag to a database entry, PwTech searches the global list of tags (comprising the tags from all entries) in a case-insensitive manner to check whether a tag with the same name already exists. If so, it will add the already existing tag with its *original spelling* to the entry. For example, if the user enters the tag name "social media" and a tag with the name "Social Media" already exists in the database (i.e., has been added previously to another entry), the tag name "Social Media" instead of "social media" will be used. Note that the tag list of each entry is unique, meaning that each tag can occur only once. Duplicates will be removed automatically when accepting the changes of an edited entry.

View options

The *View* menu provides multiple options to customize the appearance of the list view.

Show Columns ...: Select the columns to be shown in the list (see table above for the list of available fields). Note that passwords, being the most sensitive data in the database, will only be shown as plaintext if the option *Show Passwords in List* (see below) is enabled; otherwise, they will be hidden behind asterisks (***).

Show Passwords in List: Enable or disable this option to either show the passwords of the entries in the list as plaintext or hide them behind a sequence of asterisks (***). Note that the sequence is used for all entries, independent of whether the password field is actually filled or empty.

Sort Entries By ...: Select the column that you want to use as a sort criterion for the list. Alternatively, you can click on the corresponding column header in the list view to sort the list in ascending order. Clicking twice on the column header reverses the sort order (i.e., descending order). Items can be sorted in ascending (item with lowest value first) or descending (item with highest value first) order.

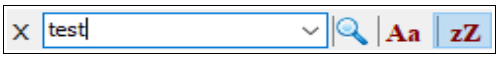
Sort Tags By ...: Tags shown in the list of tags can be sorted by *name* or *frequency* (number of entries the tag has been assigned to) in ascending or descending order. Note that the "All" field (to show all entries) always comes first, independent of the sort criterion. If sorted by name, the field "Untagged" (to filter entries that do not have any tags assigned) always comes last, whereas it is sorted normally if the list is sorted by frequency.

Change List Font: Changes the font that is used to display the list items.

Change Password Font: Changes the font that is used to display passwords in the right-hand edit panel.

Searching the database

Use the *search bar* in the toolbar to search the database:



Enter the text to find in the box and left-click on the **button** or press <ENTER> to start the search, or select an existing entry from the drop-down list. Found entries will be shown in blue color in the list view.

To clear the search text and show all entries again in the list view, press the **"X" button**. To select the fields to be searched, *right*-click on the button. Press the **"Aa" button** in a "down" or "up" state to perform a case-sensitive or case-insensitive search, respectively.

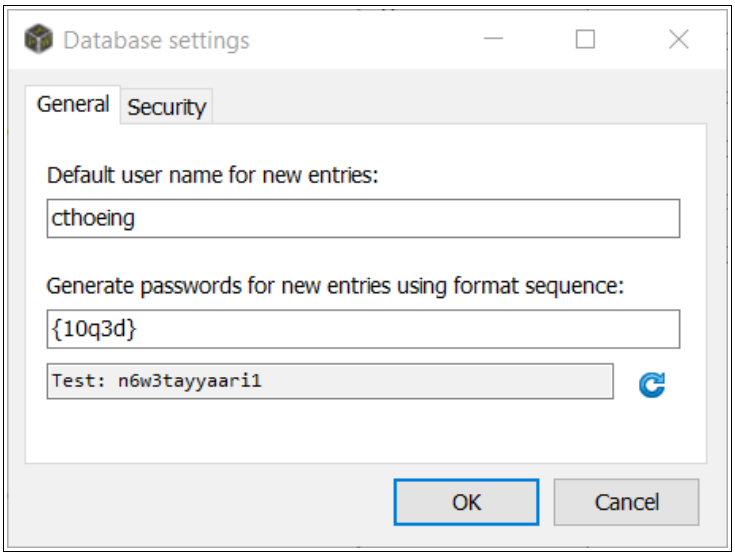
The database can also be searched in a "*fuzzy*" manner to find entries that match the text to be found only approximately. For example, the strings "Bank account" and "Bankacc" share an identity of 71%. Press the **"zZ" button** to enable fuzzy searching. Note that a fuzzy search is always performed in a case-insensitive manner, and that the first character of the search text must match accurately (for performance reasons). In the fuzzy search results, the percentage of identity p ($50\% \leq p \leq 100\%$) is displayed in the first column of the list view, and the results are sorted in descending order of identity (highest first).

Global database settings

Settings affecting the database as a whole can be accessed via the *File* menu.

To change the master password of the database, select *File* | *Change Master Password*. Note that you have to enter the old password first. A new password can only be assigned if the old password entered is valid.

Database parameters can be accessed via *File* | *Database Settings*, which will open a new window:



The **General** page exposes the following parameters:

Default user name for new entries: If defined, the *User name* field of a new database entry will automatically be set to the default user name.

Generate passwords for new entries using format sequence: When adding a new entry, a password will be generated automatically using the format sequence and displayed in the *Password* field. Press the **button** to test the format sequence and display a random password in the box below.

The **Security** page exposes the following parameters:

Encryption algorithm: Algorithm that is used to encrypt the database contents in the file. PwTech currently supports two encryption algorithms, *AES* and *ChaCha20,* which are also used for generating random data. See *Configuration* | *Security* for more details on these ciphers.

Number of key derivation rounds: Specifies the number of rounds to perform when deriving a cryptographic key from the master password of the database. The purpose of applying a key derivation function (KDF) is two-fold: First, to obtain a key that exactly matches the key size of the encryption algorithm (256-bit for algorithms supported by PwTech) without losing any entropy provided by the password, which can be of variable length. Second, to slow down guessing and dictionary attacks by applying multiple rounds of the KDF. As the KDF has to be applied for every guessed password or dictionary entry, the computational effort to find the correct password can be increased considerably. Another benefit is that the master password can easi-

ly be combined with a random *salt* (random data block of a specified size) in the KDF, which prevents attacks based on lists of precomputed keys.

PwTech uses the *PBKDF2* key derivation function with *SHA-256-HMAC* (256-bit digest size) as its underlying cryptographic primitive, and a random 256-bit salt that is stored in the database file and combined with the master password. The default number of KDF rounds is 16384, minimum is 1. It can be increased up to 4,294,967,295.

Increasing the number of KDF rounds increases the protection against guessing and dictionary attacks, but at the same time slows down the processes of opening the database and changing the master password.[3] A delay of one to a few seconds for the KDF calculation may be a reasonable compromise between protection against attacks and database performance. Press the **button** to calculate the number of rounds for a delay of 1 second.

Note that you have to re-enter the master password when changing the number of KDF rounds.

Additional functions

To perform a **drag & drop operation** for a field, click on the field *label* (e.g., *Title*, *User name*, *Password*) in the right panel, hold the left mouse button, drag the field contents to the desired location, and release the mouse button to drop the contents.

The password box can also act as a drop *target*, so that you can drag passwords from the main window or other locations to it.

3 Note that the process of saving the database is not affected, since the encryption key is precomputed when opening the database and held in memory until closing the database.

Configuration File (*PwTech.ini*)

All program settings are stored in an .ini file named *PwTech.ini*, which you can also view and modify manually in a conventional text editor. The location of this file depends on the mode in which the program is run, i.e., *portable* or *installation* mode[4].

In the portable mode, the file is located in the program folder along with *PwTech.exe*, e.g., `C:\Program Files\Password Tech`. In the installation mode, the file is located in the *%APPDATA%* folder, which is specific for each user on the computer. On Windows 7, for example, *%APPDATA%* is defined as `C:\Users\{user}\AppData\Roaming` (with {*user*} as a placeholder for the user name).

When PwTech is installed via *PwTech-xxx-Setup.exe*, the `UseAppDataPath` flag in the *PwTech.ini* file located in the installation folder is set to "1" (True) during setup, and PwTech will store all future settings in a *PwTech.ini* file located in the *%APPDATA%* folder (if the file doesn't exist, it will be created). If the flag is set to "0" (False) or doesn't exist, which is the case when PwTech is extracted from *PwTech-xxx-Portable.zip* file into a folder without any previous installations of the program, PwTech will store its settings in the program folder.

In previous versions (PWGen <2.5.1), the program could exclusively be run in the portable mode and thus stored its .ini file always in the program folder. However, this leads to problems when Windows prevents applications from writing to the `Program Files` folder for security reasons (e.g., when the program is installed with administrator rights and executed as a normal/limited user afterwards). This conflict should be solved by the introduction of the installation mode and usage of the *%APPDATA%* folder instead of the program folder for storing application-specific data. You can easily switch between both modes by editing the *PwTech.ini* file in the program folder and setting the `UseAppDataPath` flag to "0" (= portable mode) or "1" (= installation mode). If the entry doesn't exist yet, simply open the file in your favorite text editor and insert a new line into the "[Main]" section of the file:

```
[Main]
UseAppDataPath=1
```

Thus you can easily make an installed version portable by copying the program folder to a USB flash drive, for example, and changing the UseAppDataPath flag in the *PwTech.ini* file to "0".

4 This setting also applies to the *randseed.dat* file containing 32 bytes of entropy from the random pool (see: *Random Pool*).

Command Line Options

Command line switches have to be prefixed by either one or two minus characters (`-` or `--`), or by a slash character (`/`). The currently supported switches are listed in the following table. Some switches require additional arguments, which are enclosed with `{}` brackets if they are mandatory, or enclosed with `[]` brackets if they are optional.

Switch	Meaning
`ini {`*`filename`*`}`	Uses the .ini file specified by *filename* for reading and storing the settings of the program (instead of using the default .ini file, *PwTech.ini*). Include the file path in quotation marks ("") if it contains any spaces. If the file does not exist, PwTech loads the default settings on start-up and will try to create the file when the settings are to be saved.
`readonly`	Prevents any automatic writing to the disks/volumes on the computer, i.e., PwTech will not *automatically* save any data to the .ini file and the random seed file (*randseed.dat*). The user can still save the program settings by actively calling the corresponding function in the main menu. If the .ini file is not specified by the user, PwTech will load the default *PwTech.ini*; if this file does not exist, PwTech will load its default settings stored internally. The random seed file will be read on start-up but otherwise will not be modified in any way.
`profile {`*`name`*`}`	Loads the password generation profile specified by *name* on start-up. This is especially useful in conjunction with the *gen* switch.
`gen [`*`number`*`]`	Generates one password (if *number* is not specified or 1) or multiple passwords (if *number* is >1) *on the console* and then immediately closes the application. Note that this feature only works when PwTech is run from a console! To open a console window, type "cmd" into the Windows start menu, and navigate to the PwTech folder. Use `PwTech` (not `PwTech.exe`!) to call the program with the *gen* switch.

Examples:

- `PwTech -ini "C:\My Passwords\my.ini"` loads the settings from the file `my.ini` in the folder `C:\My Passwords`, and prevents any automatic writing to the system.
- `PwTech -profile commandline -gen 10` loads the profile "commandline" and then generates 10 passwords on the console. Note that the console output only works when PwTech is run from a console!

Questions & Answers

Which security level is appropriate for my password?

As always, this depends on your specific needs. Passwords corresponding to the lowest security level—for example, account passwords for not-so-important websites (there are certainly many of this kind)—should have *at least* 40–48 bits. A higher (say, "intermediate") security level is provided by 64-bit passwords. For sensitive data, a password of at least 72 bits is reasonable. To protect *really* sensitive data, the security should be at least 90 bits. Data that have to be protected for several centuries (if not thousands of years) should be encrypted with a password of at least 112, better 128 bits. Currently, it's technically infeasible to break 128-bit keys, and this statement will hold for many, many years. (In fact, it's *quite* unlikely that there will be ever found a practical way to search a 128-bit key space by brute force. If you don't trust in this assumption, go with 256-bit passwords—they are practically unbreakable.)

The following table shows the lengths of *N*-bit passwords created by different character sets and word lists.

***N* (bits)**	***A*) letters A-Z (4.7 bpc)**	***B*) upper-/ lower-case letters, numbers 0-9 (5.9 bpc)**	***C*) B + including 37 "special characters" (6.6 bpc)**	***D*) words from list with 8192 words (13 bpw)**	***E*) combination of D and B (examples)**
40	9 (=42 bits)	7 (=41 bits)	6	3 (=39 bits)	2 w., 3 ch.
48	10 (=47 bits)	8	7-8	4 (=52 bits)	3 w., 2 ch.
64	14 (=65 bits)	11 (=65 bits)	10 (=66 bits)	5 (=65 bits)	3 w., 5 ch. / 4 w., 3 ch.
72	16 (=75 bits)	12 (=71 bits)	11	6 (=78 bits)	3 w., 6 ch. / 4 w., 4 ch.
90	19 (=89 bits)	15 (=89 bits)	14 (=92 bits)	7 (=91 bits)	4 w., 7 ch. / 5 w., 5 ch.
112	24	19 (=113 bits)	17	9 (=117 bits)	5 w., 8 ch. / 6 w., 6 ch.
128	27 (=127 bits)	22 (=131 bits)	20 (=132 bits)	10 (=130 bits)	5 w., 11 ch. / 6 w., 9 ch. / 7 w., 7 ch.
256 (max.)	54 (=253 bits)	43	38 (=251 bits)	19 (=247 bits)	10 w., 21 ch.

In the table header, "bpc" means "bits per character" and "bpw" means "bits per word". The table shall give you an idea of the lengths of passwords composed of different character sets. It shows how the size of the character set or the size of the word list, respectively, influences the overall password length. It's obvious that the higher this size (i.e., the more items there are to randomly choose from), the shorter the resulting password for a given "security" in bits.

The next table shows the times to search the entire key spaces of *N*-bit passwords (given by 2^N); that is, for every *N*-bit password, it shows the time to "break" it. The speed (number of keys per second) with which this "brute force attack" can be carried out is limited by the attacker's financial resources (the amount of money the attacker can dispense), as well as by current technology. Note that neither of these variables can grow infinitely: First, limited finan-

cial resources are commonplace; second, the computer power is limited by the propagation speed of electromagnetic waves. Considering this second assessment, it becomes evident that attacks on 128-bit keys are infeasible (at least in this universe ...).

***N* (bits)**	**10^6 s^{-1}**	**10^9 s^{-1}**	**10^{12} s^{-1}**	**10^{15} s^{-1}**	**10^{18} s^{-1}**	**10^{21} s^{-1}**	**10^{24} s^{-1}**
40	12.7 days	18.3 min	1.1 s	< 1 s	< 1 s	< 1 s	< 1 s
48	8.9 years	3.2 d	4.7 min	< 1 s	< 1 s	< 1 s	< 1 s
64	5.8E5 y	584.9 y	213 d	5.1 h	18.4 s	< 1 s	< 1 s
72	1.5E8 y	1.5E5 y	150 y	54.6 d	1.3 h	4.7 s	< 1 s
90	3.9E13 y	3.9E10 y	3.9E7 y	3.9E4 y	39 y	14.3 d	20.6 min
112	1.6E20 y	1.6E17 y	1.6E14 y	1.6E11 y	1.6E8 y	1.6E5 y	165 y
128	1.1E25 y	1.1E22 y	1.1E19 y	1.1E16 y	1.1E13 y	1.1E10 y	1.1E7 y

To give you an idea of the computer power available nowadays: The network *distributed.net* broke a 64-bit key in 1757 days (searching 83% of the key space) with a rate of 10^{11} keys per second. More than 70,000 computers took part in this challenge. A similar project, aimed at breaking a 72-bit key, is still running with a current rate of $\sim 5 \cdot 10^{11}$ keys per second. Provided that the rate remains constant over the entire running time, it would still take over 200 years to search the entire key space. So, given an attacker who has access to a computer power comparable to that of *distributed.net* with a rate of 10^{12} keys per second, a 72-bit key may be considered secure. To put it another way, a 72-bit key is sufficient for protection against attackers who lack large financial resources. Now imagine an attacker who can afford a search rate which is *one million times* higher than that of *distributed.net*, resulting in approximately 10^{18} key trials per second. A 72-bit key is not sufficient here, so we should resort to a key of at least 90 bits in size. If the attacker is able to afford even more money and thus more computer power, a key size of 112 bits and more should be strongly preferred.

Of course, these incredibly big numbers for key spaces and their complexity are pretty impressive—but don't be misled by the conception that a sufficiently long, randomly chosen key automatically provides 100% security! Only megalomaniac fools try to break keys that are far beyond the scope of computer power. Clever attackers know that there are easier, faster and more effective methods to get hold of a password. Think of spyware, keyloggers, computer viruses, security flaws in operating systems, spies, double agents, truth drugs, torture... only to name a few malicious non-academic methods to find a key (you may call them "real-life brute force attacks").

Which security measures should I take when generating strong passwords?

First, you should make sure that your system is not infected with malicious software (like spyware, etc.). Before generating a "really strong" password, make sure that the random pool is "full of entropy", i.e., the "total entropy bits" counter has a value of at least 256 bits. When you copy passwords to the clipboard, clear the clipboard if you don't need them anymore (click on the *Clear Clipboard* button or select the corresponding menu item in the system tray menu). To clear the password box in the main window, just click on *Generate*, so that the contents of the box will be overwritten. To clear the password list, just close the window. PwTech automatically

overwrites the text in this list when the window is closed or when a new password list is generated. Don't save sensitive passwords *as plaintext* anywhere on your hard disk. Instead, it is recommended to encrypt them (use the text encryption utility for this purpose, see: *Tools*) and save them as ciphertext.

Last but not least, I recommend you to regularly wipe the free space on your hard disk, since it may contain remains of sensitive data which were swapped out by Windows some time ago. A first-rate tool for accomplishing this task is *Eraser*.

Is it possible to memorize those random passwords?

Yes, it is, although it may appear rather impossible on first sight. Maybe pass*phrases* composed of random words are easier to memorize for you than random characters. Try to memorize the passphrase by "visualizing" the words or by making up a funny "story" where these words play a role. Random characters can be memorized by really learning them by heart; if necessary, write them down somewhere for a *short time*, and as soon as you have succeeded in truly memorizing them, destroy the material *irreversibly*. You could also try to make up a story, the words of which begin with a character of the password. You see, lots of possibilities. So with some effort, you will eventually succeed in memorizing a strong password generated by PwTech.

In most cases, it's sufficient to memorize one or two really secure passwords (that is, having a security of at least 72 or better 90 bits); the others can be easily stored in a password safe (see: *Can I use PwTech as a password safe?*).

What about pronounceable passwords?

As of version 2.3.0, PwTech allows generating phonetic (pronounceable) passwords based on language-specific trigram frequencies; this feature is accessible by entering `<phonetic>` into the *Character set* field (see: *Include Characters*). However, note that the trigram-based password generation is restricted to languages with Latin-based alphabets (letters a–z)! Alternatively, you may generate artificial, pronounceable "words" by applying the *Format Password* option. Passwords generated by the rule `%4[%3[%c%v%] %]`, for example, will look like this: `sapifu kixezu cehiwu lukuro`. You can play around with the placeholders for consonants and vocals to create passwords that suit your needs.

Can I use PwTech as a password safe?

As of version 3.0.0, PwTech comes with the password manager *PassCube* that allows you to store passwords in a database encrypted with a master password.

Alternatively, if you prefer working with plain texts, you can use the basic text encryption function of PwTech (see: *Tools*). For editing, you may use a any text editor (such as *Notepad++*, but Windows' rudimentary *Notepad* does it, too). Whenever you need a password, for example, for an Internet service, use PwTech to create a random password of any length and store it together with the user name (or the URL as an alternative) in the safe. To "close" the password

safe, copy the whole text block to the clipboard, encrypt it with a secure master password and replace the plaintext of your password safe with the ciphertext in the clipboard.

As an another alternative or as a backup solution for a password safe, you may want to have a look at the *MP Password Generator* in PwTech, which encapsulates a "password hasher" functionality: The user provides a secret master password and a unique parameter string (such as the name of the website for which the password is to be created), and the program generates a (practically) unique password from this data. Thus, an encrypted database is not required anymore; instead, each password can be reproduced by entering the user's master password and the specific parameter. The MP Password Generator allows storing the parameter—password pairs in a PassCube database for additional safety.

Which kinds of word lists does PwTech accept?

In principle, PwTech accepts all (Unicode or ANSI) text files containing some kind of "words". These words must be separated by a line break, a space, or a tabulator. The upper limit for the word length is 30 characters (you may decrease this length down to 1, see: *Advanced Password Options*). Word lists must contain at least 2 different words. Note that PwTech does not accept duplicate words; the procedure that checks for duplicates is case-sensitive, i.e., it does *not* ignore upper-case and lower-case letters! However, you can force PwTech to convert all letters to lower-case via *Advanced Password Options*. PwTech will not add more than 1,048,576 words—upon reaching this limit, the program will stop the progress. If your word list contains more words, you can try to select shorter words by decreasing the maximum word length.

As explained before, the program accepts text files consisting of words. Here is an example which uses the words in *license.txt* to create the following passphrase:

```
original, (and number. (whether judgment
```

Note that PwTech does not remove punctuation marks or other non-letter symbols.

How to interpret the information about the random pool?

Whenever you generate "indeterministic" events by pressing keys, clicking with your mouse or by moving your mouse within PwTech's windows, the application collects "entropy" from messages sent by Windows, as well as from a high-performance counter (RDTSC processor instruction, if available; alternatively, the program calls functions provided by the Windows API; see: *High-Resolution Timer*). Additionally, the program derives entropy from several system-specific parameters, which is done in regular intervals in the background.

The actual "random pool" has a size of 32 bytes and can thus provide a maximum Shannon entropy of 256 bits. As explained in the *Step-by-Step Tutorial* above, the pool is completely filled with entropy if the progress bar is full—but the entropy counter can exceed this limit anyway. Note that this counter only serves informational purposes—it's absolutely impossible for the random pool to yield more than 256 bits of Shannon entropy! The counter just informs you about the total amount of entropy bits added to the pool, which might be of your interest if you consider PwTech's entropy estimations as too optimistic.

If you generate passwords, PwTech uses the pool contents as key for the AES (*Advanced Encryption Standard*) cipher which is then used for generating random numbers (see: *Random Pool* for more technical details). This means that a certain amount of entropy is "consumed", so to speak, from the random pool. As a consequence, the entropy counters will be decreased by the entropy content of the generated password(s). So generating five 48-bit passwords will consume 240 bits of entropy from the pool. (Of course the counters cannot fall below 0.)

Keep in mind that this loss of entropy only applies to the case that you actually *use* the generated password(s). If you discard a password and generate a new one without refilling the random pool, the entropy of the discarded password has *not* been lost. However, PwTech presumes that you use every password you generate. So once you have filled the random pool with entropy, you may effectively ignore the loss of entropy when generating passwords, as long as you don't use them. This can be explained as follows: Whenever you generate passwords or, more generally, random data using a fixed 256-bit state of the random pool, the entropy of the generated random sequence *can never exceed 256 bits*. If the apparent (!) entropy of the sequence exceeds 256 bits, it would be "easier" (in purely theoretical terms) for an attacker to find the 256-bit state of the random pool than to find the actual random sequence originating from this very 256-bit state. To put it another way, each 256-bit data block within the sequence has an *independent entropy of 256 bits* when treated separately from the rest, but adding more data from the same sequence to a 256-bit block doesn't increase its security. This does *not* mean that only the first 256 bits of random output are secure and the further output is deterministic/insecure and thus useless. It simply means that you cannot increase the entropy of the random output beyond 256 bits. *Theoretically*, an attacker could predict the output following the first 256 bits, *provided that he is able to break the 256-bit state of the random pool*. However, as emphasized multiple times within this manual, breaking 256-bit keys is practically unfeasible by any standards (well, neglecting the exploitation of time travels, black holes, parallel universes, etc. here...).

Note: PwTech uses additional time stamps, independent of the 256-bit state of the random pool, when generating random numbers, which can increase the entropy up to 384 (256+128) bits. Nevertheless, since such security-relevant estimations should be made in a rather conservative way in the realm of cryptography, the internal limit is set to a fixed value of 256 bits.

Technical Details

Random Pool

Whenever you interact with your computer—for example by pressing a key, by clicking with your mouse or by moving your mouse—you create an event that is, unlike most internal computer events, to a certain degree indeterministic or unpredictable. The term "indeterministic" means here that these events can afford a certain amount of randomness—and let it be just a few bits. (Due to their random character, indeterministic events are sometimes—rather loosely—referred to as "*entropy*"; this is not to be confused with the entropy term in thermodynamics!) By calling a high-performance timer (like the RDTSC processor instruction available on all modern CPUs, see: *High-Resolution Timer*) when the user generates such an event we can efficiently make use of its random character. Moreover, the operating system Windows provides additional (and again partly indeterministic) information such as the type of Windows control where the message was sent to, the cursor position and the (low precision) date and time when the message was sent.

We can gather all this entropy from user-generated events in a so-called *random pool*. At this point it is essential to note that this data does not have to be purely random—it is sufficient that it contains some bits of uncertainty (the more, the better, of course). The "trick" is now to apply a *cryptographic hash function* to this entropy in order to "distil" randomness from this data. This operation yields data that looks truly random, although it cannot contain more information than the original partly-random data. If we now collect entropy amounts large enough to provide (in total) sufficient uncertainty, we can generate sequences that do not only look random, but in effect *are* random. For example, to generate a highly secure 128-bit password, we fill the random pool with enough user inputs (i.e., entropy) first—by entering some text on the keyboard, or by moving the mouse, etc. Then we use this data to distil the required 128 bits of truly random data from the pool and convert it to a secure password. This is the basic concept of random number generation in PwTech. The "real implementation" is a bit different.

In mathematical terms, the process looks as follows: HMAC-SHA-256(K, M) is used as secure hash function. The HMAC algorithm uses an additional key K to compute the 256-bit message-digest of a message M. (Note that in this case, the "key" does not have a special purpose here; we simply use the HMAC algorithm instead of the plain hash function because the former is thought to have better "randomness extraction" properties compared to the latter.) To distil randomness and to generate a new pool, the old pool contents and the entropy data are processed in the following way:

$$P \leftarrow HMAC(P, S \,||\, T), \tag{1}$$

where P is the random pool (32 bytes), *HMAC* is the HMAC hash function, S is the entropy data of any length, and T is a time stamp from a high-resolution timer. "||" denotes simple concatenation of the sequences. PwTech uses a memory block of a defined length to buffer subsequent entropy inputs; when this buffer is full, it will be hashed according to eqn. (1) and cleared (overwritten with random data) afterwards.

To generate cryptographically secure random numbers, PwTech uses the AES (*Advanced Encryption Standard*) encryption algorithm in counter (CTR) mode. AES is set up with a 256-bit key and operates on 128-bit blocks. In order to derive a key *K* for AES, the pool is first updated (eqn. (1)), then the pool is hashed:

(2) $K \leftarrow HMAC(T, P)$.

This procedure ensures that *P* cannot be derived from *K*, and vice versa. Random numbers are then generated by encrypting a 128-bit block counter *C* (3a), which is incremented by 1 afterwards (3b):

(3a) $R \leftarrow E_K(C)$,

(3b) $C \leftarrow C + 1$.

C is initially derived by encrypting two 64-bit time stamps. *R* is the next 128-bit random output, and E_K is the encryption function with key *K*. As the attacker neither knows *K* nor *C*, he cannot predict the next number *R*, even if he knows the entire previous sequence. Furthermore, *K* and *C* are changed regularly to ensure that, even in case of a *state compromise* (leakage of sensitive data in *K* and *C* to the attacker), the attacker cannot reconstruct those random blocks generated before the key change ("forward secrecy"). Using time stamps when updating the pool and when generating secret counter values *C* adds to the security of the construction, since they provide a certain degree of unpredictability.

All sensitive data (random pool, entropy buffer, AES key, counter, …) are held in RAM to prevent it from being swapped out to the hard disk. The pool is made indistinguishable from random by clearing all buffers with random data after use. This will make it difficult to locate the pool in RAM by means of any easily identifiable sequences.

For performance reasons, PwTech delays the recognition of mouse movements. For every mouse input, PwTech counts a maximum of 10 bits of entropy (max. 8 bits for the timer plus 2 bits for cursor coordinates); for every keystroke, PwTech counts a maximum of 9 bits (1 bit for the key code). In addition to these user inputs, PwTech collects entropy from various system parameters every 15 seconds, which yields 24 bits of entropy in PwTech's estimations. Note that these estimations are rather conservative—the "real" quality of these entropy sources is likely to be higher.

In order to preserve the entropy collected during runtime, PwTech writes a random seed file (*randseed.dat*) in the application folder. This 32-byte file is read on start up, its contents are fed into the random pool, and the seed file is immediately overwritten with new data afterwards. PwTech updates this file in regular intervals while running as well as on exit. If you don't want PwTech to write this file, you can use the */readonly* command line switch (see: *Command Line Options*).

Text Encryption

The text encryption can be divided into five steps: UTF-16 to UTF-8 Unicode conversion, Text compression, HMAC generation, encryption and base64 conversion.

1) The UTF-16-encoded text as received from the Windows API is converted to **UTF-8 encoding**.

2) The text is **compressed** using the LZO1X-1 compression algorithm.

3) The **256-bit HMAC** of the compressed text is generated.

4) The compressed text is **encrypted** using AES with a 256-bit key.

5) The encrypted text (ciphertext) is converted into "readable" characters (**base64** character set).

Let's start with an overview of the encryption procedure: To derive the 256-bit key from the password provided by the user, a version of the key derivation function *PBKDF2* is used, which applies the hash function HMAC-SHA-256 to the user password along with a randomly chosen 128-bit initialization vector (IV; also called "salt" in this context). This process is repeated many times (8192 iterations) in order to make password cracking much more difficult.

The text which is to be encrypted consists of a header structure (3-byte "magic" identification string, version number and length of the uncompressed text) and the compressed text (variable length). This assembly is encrypted in CBC (*Cipher Block Chaining*) mode in order to hide patterns in the plaintext:

$$(1a) \quad C_i \leftarrow E_K(P_i \oplus C_{i-1}).$$

C_i describes the ith block of ciphertext (i.e., the encrypted text) and P_i the ith block of plaintext (i.e., the unencrypted text). C_0 is the IV which is also used in key derivation. Again, E_K is the encryption function with K as key. The symbol "$\oplus$" denotes an "exclusive-or" bit operation. In decryption mode, eqn. (1a) can be reversed in a simple way:

$$(1b) \quad P_i \leftarrow D_K(C_{i-1}) \oplus C_{i-1},$$

where D_K is the decryption function.

Before encrypting the plaintext, a so-called HMAC (*keyed-Hash Message Authentication Code*) of the text is calculated. In general terms, the HMAC can be considered as a message digest of the ciphertext with K as parameter. It serves to check the decryption process in two ways: 1) *Is the key correct?*, and 2) *Has the ciphertext been altered in any way (data integrity)?* Note that PwTech cannot distinguish between these questions, i.e., it can only check if the HMAC is correct or not, but it cannot tell you on the basis of the HMAC if the decryption failed because of a wrong key or because of a corrupt ciphertext. As the attacker doesn't know the key, he cannot compute valid HMACs himself; in particular, he cannot modify the ciphertext and then make any valid modifications to the HMAC due to the strong non-linearity of the HMAC algorithm. Any modifications of the ciphertext will be recognized during the decryption process. PwTech uses HMAC-SHA-256 to generate a 256-bit HMAC of the entire plaintext block. This 32-byte sequence is appended to the ciphertext after encryption.

In the last step, the binary encrypted data generated in the previous three steps are converted to the base64 format (a character set consisting of 64 "readable" characters: mixed-case let-

ters, numbers, and the characters +, / and possibly =) in order to make it "readable" by text editors, e-mail programs, etc. In the encoding scheme used by PwTech, lines are wrapped after 76 characters to allow for a smooth display of the text in editors. (Some editors, such as Windows' *Notepad*, may crash [or take a *very long* time] when the user attempts to paste very long texts consisting of only one single line. Therefore, it seems to be prudent to create multi-line texts with shorter but more numerous lines.)

The decryption procedure is essentially the reverse of the encryption procedure. There are several "checkpoints" in this procedure, which serve to verify the validity of the key ("*Did the user enter the correct password?*") as well as data integrity ("*Is the ciphertext still in its original state or has it been modified in any way?*").

1. The ciphertext is converted from base64 (6-bit) characters to 8-bit bytes. **Check #1:** If there are illegal characters in the text, or if the resulting text is shorter than 64 bytes, or if the text length is not a multiple of 16, the ciphertext structure is invalid, which means that the text is either corrupted or not encrypted by PwTech.

2. The first 16 bytes of the ciphertext are decrypted and the "magic string" in the header structure is checked. **Check #2:** If the decrypted identification string and the default string do not match, then either the key is wrong, or the beginning of the ciphertext (first 32 bytes) has been modified.

3. The remaining ciphertext is decrypted to give the compressed plaintext.

4. The HMAC is generated and checked. **Check #3:** If this verification fails, either the key is wrong, or there were modifications *somewhere* within the entire ciphertext. However, as the verification of the identification string in the header must have succeeded in step (2), it is actually more likely that the ciphertext has been modified.

5. Finally, the plaintext is decompressed and converted from UTF-8 to UTF-16 to give the original plaintext. [**Check #4:**] Note that the decompression and the Unicode conversion can *theoretically* fail, although this should *never* happen under "normal" circumstances. If this error really occurs, the most probable cause is a heavily manipulated ciphertext, i.e., a text that has *not* been generated by PwTech.

CAUTION: *Always check the correct decryption of the ciphertext before discarding the plaintext!*

High-Resolution Timer

If possible, PwTech uses a high-resolution timer (HRT) the value of which is added to the pool when you press a key, move your mouse, or click with your mouse. It is also used when generating cryptographically strong random numbers to increase the security of the generator. Due to the high resolution of this timer every value possesses a certain degree of unpredictability, particularly when measuring the timer intervals between user-generated events such as keystrokes and mouse clicks. Therefore, it can provide a relatively high amount of entropy. Moreover, consider the following worst-case scenario that an attacker got hold of the entire state of the random pool ("*state compromise*") and thus could theoretically predict the entire sequence

of random numbers that will be generated. Adding the HRT value before generating the next random sequence offers a certain protection against this scenario: The attacker can read the timer value himself, of course, but it is actually very unlikely that he reads the same value due to the high time resolution of the timer (typically nanoseconds on modern CPUs). Since the HRT is available very quickly, it is used in every critical situation of the random pool, i.e., when updating the pool and when generating random numbers.

PwTech can use the following timers (listed with descending priority):

1. **Time stamp counter (RDTSC instruction):** The RDTSC processor instruction returns the number (64-bit) of cycles since reset. The time resolution therefore depends on the clock rate of the processor: On a 3 GHz processor, for example, each cycle takes ~0.3 nanoseconds. The RDTSC instruction is present on all x86 processors since the Pentium and provides an excellent high-resolution, low-overhead way of getting CPU timing information.

2. **QueryPerformanceCounter (Windows API):** According to the *Windows SDK*, this function returns the current value (64-bit) of the high-performance counter. This is probably just a wrapper for the time stamp counter on most systems, but the return value may be different on multicore computers. Calling this function is slower than executing RDTSC.

3. **Low resolution timer (Windows API):** Alternatively, if neither RDTSC (1) nor QueryPerformanceCounter (2) are available, PwTech retrieves the current date and time of the system. The resolution of this timer is only in the millisecond range and thus *much* lower (by several orders of magnitude) than that of the high-resolution timers (1) and (2), but may still be sufficient for measuring keyboard and mouse events.

On start up, PwTech checks if the RDTSC instruction (1) is available, which should be the case on all modern computers (Pentium processor onwards). If this check fails, PwTech tests QueryPerformanceCounter (2). If both tests fail, PwTech uses the current system date and time (3). However, if (1) or (2) are allegedly available, PwTech briefly checks if they can really hold the high resolution; this test might fail when PwTech runs on emulated systems, for example, and if it fails, PwTech uses (3) instead.

Note that PwTech constantly checks the resolution of the timer values and decreases the entropy rating if the resolution is low (i.e., in the millisecond region). Although the security of the random generator does *not* entirely depend on the entropy of the timer values because a lot of other parameters are additionally incorporated into the pool, it is strongly recommended to run PwTech on systems where a high-resolution timer (1 or 2) is present due to the higher entropy and due to the increased security in critical situations of the random generator. You can check which timer is used by selecting the item *Timer Info* from the *Help* menu.

Contact & Further Information

Contact

For bug reports and general discussion, please use the ticket system and forum, respectively, provided by Sourceforge.net. You can also contact me via e-mail: `c.thoeing@web.de`

Translations

Translations of PwTech into languages other than English and German are always very welcome. For a list of translations that are available so far, see here. If you want to update an existing translation file or create a new one, I recommend using the PwTech Translation Utility that I have created specifically for this purpose. This program uses the file *German.lng* (which accompanies every release of PwTech) as a template for new translation. Please send me your translation via e-mail, so that I can publish it on PwTech's project page.

Word Lists and Trigram Files

Have a look at the PwTech project page to access additional word lists and trigram files suitable for generating passphrases and phonetic passwords, respectively. If you have created a word list or trigram file and want to share it with other users, please send me the file, so that I can publish it on the web. Alternatively, you can find special word lists in several languages on the "Diceware" homepage; however, you may want to extract the actual words (without the numbers/indices before the words) first before using the "Diceware" lists in PwTech.

Please Donate!

As you can certainly imagine, developing and maintaining a software project like PwTech—even though it may appear rather small—requires a lot of effort and commitment, especially if you have other responsibilities in your regular job as well as at home. If you like PwTech and use it frequently, please donate to the project. Your donations will encourage me to further develop the application and keep it bug-free. For example, one of the future goals is to provide a 64-bit version of the software. *Thank you!*

If you donate at least one of the following amounts, you will receive a so-called **Donor Key** that can be used to change your PwTech edition from *Community* to *Donor* (via *Help* | *Enter Donor Key*), and thus deactivate the "Please-donate" message boxes shown with a certain frequency at the start of the program. Also, donors will get **priority support** via e-mail.

Donate at least 9 EUR / 10 USD:	Activate **Donor** edition that is valid for the current version of PwTech and three future updates.
Donate at least 18 EUR / 20 USD:	Activate **Donor Pro** edition that is valid for the *entire lifetime* of PwTech version 3 (current version including *all future updates* for major version 3).

The key consists of 16 characters and will be sent via e-mail (usually within 24 hours) after you have made the donation via PayPal. The key contains an 8-character *Donor ID* that will be displayed in a message box after entering it, as well as in the *Help* | *About* window.

Acknowledgment

Thanks to Embarcadero, Inc., Brainspark B.V., Markus F.X.J. Oberhumer, Arnold G. Reinhold, Aha-soft, Dominik Reichl, Sami Tolvanen, Markus Hahn, Theodore Ts'o, Robert J. Jenkins, and Wei Dai for code & inspiration! Thanks to all the users for their donations, helpful comments and suggestions!