

Introduction

Advanced ZIP Password Recovery (or **AZPR**) is a program to recover lost or forgotten passwords to archives (compressed files) created with ZIP archivers (PKZip, WinZip, PowerZip etc).

At the moment, there is no known method to extract the password from such files, so the only available methods are "brute force" and dictionary attacks; in some cases, known-plaintext attack is also available.

The key features of **AZPR** are:

- The program has a convenient **user interface**
- The program is very fast: brute-force attack speed is up to **fifteen million passwords per second** on modern CPUs like Pentium III
- The program can work with archives containing only one encrypted file
- Archives created by various software packages are supported
- **All compression methods** are supported
- **Self-extracting archives** are supported
- The program is **customizable**: you can set the password length (or length range), the character set to be used to generate the passwords, and a couple of other options
- You can select the custom character set for brute-force attack (non-English characters are supported)
- **Dictionary-based attack** (with word mutations) is available
- The "brute-force with mask" attack is available
- Very fast and effective **known-plaintext attack** is available
- The maximum password length is not limited (in registered version)
- No special virtual memory requirements
- You can interrupt the program at any time, and restart from the same point later
- The program can work in the **background**, using the CPU only when it is in idle state

The next versions will have much more useful features, of course.

Please note that password encryption in ZIP is relatively strong – for example, look at [Application Notes for PKZIP](#) for details about ZIP encryption). If the password is long enough and well selected, and known-plaintext attack is not applicable, there is no way to recover it in a reasonable time. The "practical" limit is about 10 characters.

Requirements

- Windows 95 (any version), Windows 98, Windows ME, Windows NT 4.0, Windows 2000 or Windows XP running on Pentium CPU
- about 4 megabytes of free memory (for known plaintext attack – about 34 megabytes)
- about 1 megabyte of hard disk space
- patience...

About ZIP passwords

Here is a part of **PKZIP format specification** available from [PKWARE](#):

The encryption used in PKZIP was generously supplied by Roger Schlafly. PKWARE is grateful to Mr. Schlafly for his expert help and advice in the field of data encryption.

PKZIP encrypts the compressed data stream. Encrypted files must be decrypted before they can be extracted.

Each encrypted file has an extra 12 bytes stored at the start of the data area defining the encryption header for that file. The encryption header is originally set to random values, and then itself encrypted, using three, 32-bit keys. The key values are initialized using the supplied encryption password. After each byte is encrypted, the keys are then updated using pseudo-random number generation techniques in combination with the same CRC32 algorithm used in PKZIP.

If nothing from the above text is very clear to you, don't worry – **AZPR** will do its work anyway. We've included this description to point out that the password is not stored anywhere in the ZIP file, and so it cannot be just extracted or decrypted. Instead, we can try to "guess" it by trying different passwords: all possible combinations in a given range, or from a wordlist, etc. No, **AZPR** doesn't try to unzip the whole file (with a given password) to verify that this password is valid; there are some very fast ways to check possible password validity, especially on modern CPUs. **AZPR** can test a few million passwords per second, and so the likelihood of finding a valid one is very high. There is still no guarantee that the password will be recovered, but here the human factor plays its role: most people use short and/or easy to remember passwords. We estimate the success rate as 90-95%.

The only exception is the "known-plaintext" attack. In most cases, it doesn't recover the password, but allows to get the three 32-bit keys mentioned above; using these keys, the archive can be decrypted so you will not need the password to get in. Unfortunately, this attack is not always applicable.

Password-encrypted file

Just enter the name of the ZIP archive you'd like to get the password for. Use the "Browse" button (or F3 key) to select it, or press the "recent files" button (with a small down arrow) to pick from the list (if you've used **AZPR** on your target archive before). Alternatively, you can use drag'n'drop – just drag the file (with a mouse) from Windows Explorer, and drop it to the **AZPR** window. If all the settings are correct, the attack will be started immediately.

Type of attack

Brute-force or dictionary attack. Brute-force, Mask, Dictionary and Plaintext attacks are available.

Brute-force range options

Instructs the program what characters have been used in the password. You can choose from all capital letters, all small letters, all digits, all special symbols and the space, or all printable (includes all of the above). The special characters are:

!@#\$%^&*()_+-=<>./?[]{}~:;`'|\"

Alternatively, you can define your own character set (charset). Just mark the "User-defined" checkbox and click on "Custom charset..." (at the right of the option). In the input window, enter all chars of your password range; for example: if you remember that your password was entered in the bottom keyboard row ("zxcv...") - your password range should be "zxcvbnm,./" (or in caps: "ZXCVCBNM<>?"). You can also define both of these: "zxcvbnm,./ZXCVCBNM<>?". In addition, you can load and save custom charsets, or combine them using the "Add charset from file..." button.

Just a note about "Convert to OEM encoding" option in the "User Defined Charset" option. Be sure to select it if the password contains any non-English characters, and the archive has been created by a DOS-based compression utility (like PKZIP, for example). Otherwise, the password will not be found.

Start from password

This option may help, for example, if you know the first character(s) of the password. For example, if you're sure that the small letters have been used (from 'a' to 'z'), the length is 5, and the password definitely starts with 'k', than type 'kaaaa' here. Please also note, that if you press the "Stop" button when **AZPR** is working, the program writes the current password to this window ("Start from password"). It can be used later to restart the program from the same point.

Please note that the program verifies the passwords according to the following character order:

- CAPITAL letters: 'A'..'Z'
- the space
- small letters: 'a'..'z')
- digits: '0'..'9'
- special characters: !@#\$%^&*()_+-=<>./?[]{}~:;'|\\"/>

You can also use **End at** field to set the password **AZPR** should stop at. It might be useful if you attack the same archive on a few computers, and so can split the whole password range onto a few parts.

Password mask

If you already know some characters in the password, you can specify the mask to decrease the total number of passwords to be verified. At the moment, you can set the mask only for fixed-length passwords, but doing this can still help.

For example, you know that the password contains 8 characters, starts with 'x', and ends with '99'; the other symbols are small or capital letters. So, the mask to be set is "x?????99", and the charset has to be set to *All caps* and *All small*. With such options, the total number of the passwords that **AZPR** will try will be the same as if you're working with 5-character passwords which don't contain digits; it is much less than if the length were set to 8 and the *All Printable* option were selected. In the above example, the '?' chars indicate the unknown symbols.

If you know that the password contains an occurrence of the mask character '?', you can choose a different mask character to avoid having one character, '?', represent both an unknown pattern position and a known character. In this case, you could change the mask symbol from '?' to, for example, '#' or '*', and use a mask pattern of "x#####?" (for mask symbol '#') or "x*****?" (for mask symbol '*'). Select the mask symbol on [Advanced Options](#) page.

Password length

This is one of the most important options affecting checking time. Usually, you can check all 4-character (and shorter) passwords in a few minutes; but for longer passwords, you have to have patience and/or some knowledge about the password (including the character set which has been used, or even better – the mask).

The minimum length cannot be set to a value greater than maximum length, of course.

If the minimum and maximum lengths are not the same, the program tries the shorter passwords first. For example, if you set minimum=3 and maximum=7, the program will start from 3-character passwords, then try 4-character ones and so on – up to 7. While **AZPR** is running, it shows the current password length, as well as the current password, average speed, elapsed and remaining time, and total and processed number of passwords (Program status). All of this information except average speed and elapsed time, which are global, is related only to the current length.

Dictionary options

Simply select the desired dictionary file. In addition, you can select an option *Smart mutations* or *Try all possible upper/lower case combinations* – it may really help if you're not sure about the register the password has been typed in. For example, let's assume that the next word in dictionary is "PASSWORD" (the case, actually, doesn't matter here). With the second option enabled, the program will just try all possible combinations, like:

```
password
passworD
passwoRd
passwoRD
passwOrd
...
PASSWORD
PASSWORD
```

However, checking all such combinations takes a lot of time: in the example above, **AZPR** will check 2^8 words (i.e. 256) instead of one. With smart mutations, you can eliminate a number of "virtually impossible" combinations, and here are all the words which will be checked:

PASSword	(as is)
passWORD	(reversed)
password	(all lower case)
PASSWORD	(all upper case)
Password	(first uppercase, rest lowercase)
pASSWORD	(first lower case, rest uppercase)
PaSSWoRD	(elite: vowels in lc, others in uc)
pAsswOrd	(noelite)
PaSsWoRd	(alt/1)
pAsSwOrD	(alt/2)

So, it makes only 10 combinations for each word.

The *Start line #* option allows you to start an attack from a given line (in the dictionary); if you interrupt the attack, the "current" line number will be written there (and saved to the project file, of course).

The *Convert to OEM* encoding option can be used if: the dictionary is in ANSI coding, but the ZIP archive has been created with a DOS archiver (like PKZIP), and so the actual password is in OEM coding. Changing that option doesn't make any difference if all the words in the dictionary contain latin letters only.

A small but very effective dictionary is included into **AZPR** distribution: *english.dic* (about 27,000 words). Some other very good dictionaries are available at:

<ftp://sable.ox.ac.uk/pub/wordlists/>
<ftp://ftp.cdrom.com/pub/security/coast/dict/wordlists/>
<ftp://ftp.cdrom.com/pub/security/coast/dict/dictionaries/>

Also, please have a look at our **Password Recovery Software** page – you'll find a few dictionaries, wordlists and dictionary generators there, as well as the links to related sites:

<http://www.elcomsoft.com/prs.html>

Known plaintext attack

Introduction

ZIP files have a strong encryption algorithm. First, the password isn't stored anywhere in a password-protected archive. The ZIP archiver converts the password you've entered into three 32-bit encryption keys, and then uses them to encrypt the whole archive. Because of this, the total complexity of the ZIP attack is 2^{96} , i.e., we would have to try all possible key combinations. This is really a lot – even using all the computers in the world, it is not possible to check all of them, unfortunately... However, this algorithm isn't as strong as the DES, RSA, IDEA, and similar algorithms. One of the ways of breaking ZIP protection is using known-plaintext attack. If you're interested in the details of attack, find the paper "A Known Plaintext Attack on the PKZIP Stream Cipher" by Eli Biham and Paul Kocher. **AZPR**'s implementation of plaintext attack is very close to that paper, with some minor modifications.

Having an encrypted file created by the ZIP archiver, and the same file in unencrypted form, we can make some calculations and retrieve the encryption keys used to protect that file. Usually, a ZIP archive contains several files and all of them have the same password (and therefore the same encryption keys). This means that if we get the encryption keys for one of these files, we'll be able to unprotect all the others! Furthermore, it won't take as much time as trying all possible combinations of encryption keys. To perform plaintext attack, all you need is one file from the archive, compressed by the same archiver and by the same method as an encrypted one.

Selecting the correct archiver is a bit complex, however; unfortunately, the ZIP file format doesn't contain any data which might help to identify the archiver. In fact, you may need to try several archivers (of course, only if you don't remember which particular utility you've used). A good check that the plain file is correct is the size difference between it and the encrypted file: the encrypted file must be exactly 12 bytes larger. Also, the files must have the same CRC and uncompressed sizes. **AZPR** automatically checks these conditions for selected files, so all you need to do is to create a "plain" ZIP archive.

Description

To perform plaintext attack you need to:

- Find an unencrypted file which also exists in the password-protected archive.
- Compress it with the same method and the same ZIP archiver as used in the encrypted archive. Note that this is required because **AZPR** checks file sizes and file checksums. (You can, however, use plaintext attack on a partial file; see the description below).
- Run **AZPR**, select encrypted archive, then select "plaintext" attack and browse for archive with unencrypted file.

After that, **AZPR** will check the files, and if there are matching ones, the attack is started.

There are two stages in "plaintext" attack, plus two password search additions (note that timings are estimated for Intel Celeron working at 366 MHz):

1. **Keys reduction cycle.** At this stage, **AZPR** needs about 34 megabytes of (virtual) memory. This cycle takes from one to three minutes (depending on the size of the plaintext). If you haven't got enough physical memory, it may take a bit more time. After this stage, **AZPR** will free most of that memory and work with only 2-4 megabytes. Please also note that the time required to complete this stage cannot be estimated, and so for the first few minutes the progress indicator will read 0%, after that it'll start to increase rapidly.
2. **Searching matching keys.** This is the main stage of "plaintext" attack. Now you can see how much time you need (worst case) to recover the archive. Depending on the size of the plaintext, this stage can take from 5 minutes to several hours. At that stage, you can stop the attack at any time without risk; the program will write a resume value into the *Start from* field (and save it into the project file, of course). Note that the first stage (keys reduction cycle) will be performed again upon resuming (but it only takes a few minutes).

When **AZPR** finds valid keys, it tries to find the password correlated with them. Due to some reasons, the password search can be easily done for 9 characters long (and shorter) passwords with any symbols, and passwords with up to 10 printable symbols – during a couple of minutes.

If **AZPR** can retrieve the password, it'll display the standard statistics message with it, if it can't – with encryption keys only. Please note that in most cases you don't need the original password because having encryption keys you can easily decrypt the ZIP archive so it will not require the password to unzip it.

Attack on partial file

Sometimes ZIP archives (where the one is password-protected and the second isn't) may differ in size. For example, WinZip can create such ones if the source file almost cannot be compressed. Encrypted files has 12 bytes at least, so when WinZip starts the compression routine, it may select another method to keep compression ratio good. But note that it is very unusual case. However, you can perform plaintext attack on such files anyway – just keep in password-protected archive only one file (that will be attacked); of course, backup your original files first. And keep only one file in "plaintext" archive as well. Run the attack, and **AZPR** will ask for

confirmation for “partial” attack. Click ‘Yes’ and select the number of bytes to use as plaintext. Because we don’t know how many bytes can be the same, it’s good idea to start from 1-3Kb (it most cases it’s enough) and decrease this number if **AZPR** won’t be able to find encryption keys.

Current version notes

1. "Plaintext" file must be at least 12 bytes long.
2. "Plaintext" attack can be saved on the second stage only; after restarting, the first stage will be performed (again) anyway.
3. No time estimation for the first stage. But you can expect that it'll take a few minutes.
4. In any case, you need about 34 megabytes of RAM. If you don't have so much RAM, you need enough space on the for swap file on the disk (and patience – using virtual RAM will greatly decrease the performance). So, we recommend to use the “known plaintext” attack with at least 40-48 megabytes of RAM.

Test results

Here are the results (benchmarks) of "known plaintext" attack for the different files (on Intel Celeron 366MHz with 64MB RAM).

<u>File size (bytes)</u>	<u>Stage #1 time</u>	<u>Stage #2 time</u>
16	20s	2d 12h
32	33s	8h 30m
64	38s	3h 30m
128	45s	1h 45m
256	52s	42m
512	52s	20m
1024	52s	8m
2048	1m 5s	5m 30s
4096	1m 5s	4m
8192	1m 14s	4m
16384	1m 30s	4m
32768	2m 10s	4m

Auto-save

If you'd like **AZPR** to save its state periodically, please check the appropriate option, and select the time (in minutes) between saves. If you do that, **AZPR** will create and periodically update a restore file named "~azpr.azr" (that's the default – you can change it) in the same folder where your archive is located (also by default; you can select any other folder to save that file to). This file is similar to one created when using the "Save setup" button. Even if your computer stops responding (or if power fails), you'll be able to restore breaking the password from the last saved state. Instead of using the default settings (the name of the file and the folder it will be saved to), you can also select your own settings. Enabling this option is **strongly recommended**.

Other options

Priority: background or high. If you want to start **AZPR** as a "background" process, which will work only when the CPU is in an idle state, you may select "Background". If you want to increase performance, select "High", but be aware that this will decrease the performance of *all other* applications running on your computer.

Minimize to tray: if this option is enabled, the program window will disappear from the Windows desktop when you press the "minimize" button in the top-right corner of the window (or you select an appropriate item in the system menu). The small icon will be created in the "tray" area of the task bar (near the system clock). Just double-click on that icon to restore the window.

Log to azpr.log: when enabled, the program saves all information displayed in the status window into the log-file (*azpr.log*).

Progress bar update interval: allows to set an interval (in milliseconds) between progress bar and status window updates; the default is 500 (a reasonable value). By selecting the higher value (3000, for example), you can get slightly better recovery speed.

Register: press this button to register your copy of **AZPR** (if you've got the registration code already, of course).

Update: press this button (when you're connected to the Internet) to see if there a new version of **AZPR** on our site. Note: the program uses Microsoft Internet Explorer proxy settings.

Language: the program has multi-language interface. Just select the appropriate language from the drop-down box. English is the default.

Advanced options

Use known start of the file for stored archives (hex): if your archive contains only one encrypted file, and this file is stored (i.e. not compressed), using that option is a solution to get much better recovery speed. But you have to know from 1 to 4 bytes this file starts from. There are a lot of well-known signatures, though: for example, 'MZ' (hex: 4D 5A) for executable files, 'PK' (hex: 50 4B 03 04) for ZIP files, D0 CF 11 E0 (hex) for OLE compound documents (like MS Word/Excel files) etc.

Always use WinZIP optimized attack engine if probability is greater than XX%: if your archive has been created with WinZIP (or other Windows-based ZIP tool based on the same sources -- there are many such tools) and contains at least five encrypted files, there is some good news: the speed of brute-force attack can be about three times better! **AZPR** tries to recognize such a situation automatically, but unfortunately, a ZIP file doesn't store any information about the archiver. So the program calculates the "probability" value (it depends on the number of files in the archive and other factors). If it is greater than 50%, **AZPR** suggests you to use this (optimized) attack each time you start the recovery process. You can set this option (selecting the appropriate percentage as well) for your convenience, so the optimized engine will (or will not) be used automatically. 85% is the good value to use, but you can set a higher value, if you're not sure.

Mask symbol: used for Mask attack.

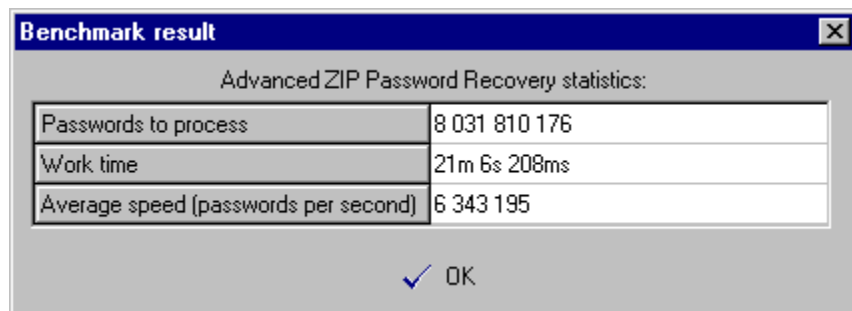
Save and Read setup

You can save your current **AZPR** setup into a specified file (with extension AZR). When you press the "Save setup" button (or F2 key), the "Save file" dialog appears. Just select a file name (e.g. "myarch.azr"), or select an existing AZR-file for overwriting. You can read your setup later – simply press the "Read setup" button.

Alternatively, you can use drag'n'drop – just drag the previously saved azr-file (with a mouse) from Windows Explorer, and drop it onto the **AZPR** window. If all the settings are correct, the attack will be started immediately.

Benchmark

If you would like to estimate how long the Brute-force or Mask attack will take, or test **AZPR**'s speed on a particular archive, use the benchmark feature. Just select all the desired options, then press the **Benchmark** button (next to **Stop**). The program will work for about 10 seconds, and display some statistics afterwards:



Here you can see the total number of passwords (according to the options you set), average program speed, and estimated time. Please note that real time might be slightly different, because the speed of the program depends on how many other applications are running at the same time.

Recovering process

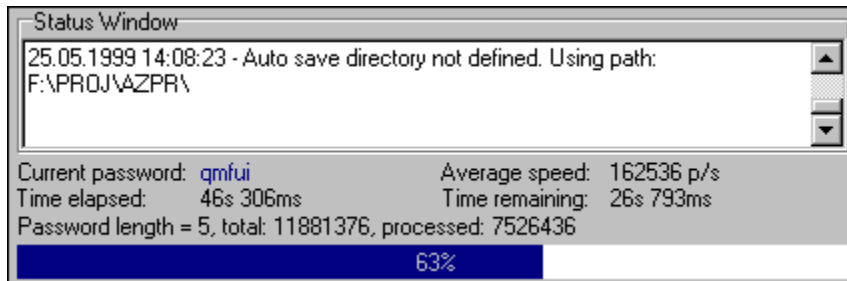
When all of the options are selected, all you have to do is press the **Start** button on the toolbar (or **F9** key) and wait. During the attack, you'll be able to see the Program status – number of passwords already tried, elapsed and estimated time, etc.

Please note that you can stop the recovering process at any time (**Stop** button or **F10** key), to continue it later (or just save the project). Consult the Start from password and Save and Read setup chapters for further details.

In **known plaintext** attack, you can stop the process at any time as well, but resuming is possible only if you do that on second stage (“searching for keys”) only, but the first stage should be performed again anyway. The first stage takes a few minutes, however (in contrary to the second one, which may take 2-3 days in some cases). Note: resuming known-plaintext attack is available in registered version only.

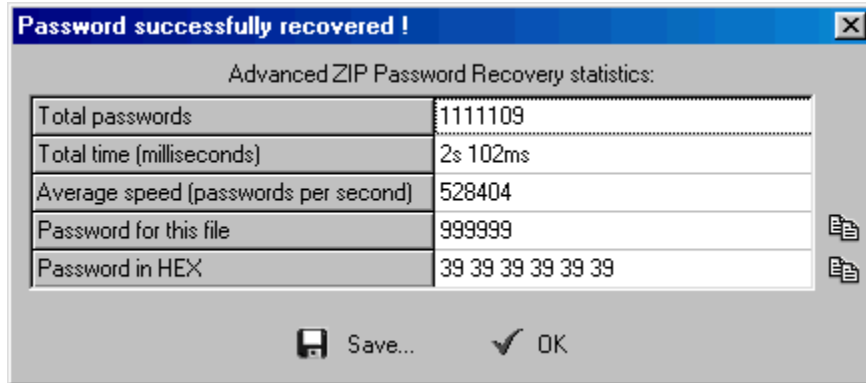
Program status

When the recovering process is in progress – the current password, average speed, elapsed time, remaining time, total number of password of given length, and number of passwords already processed are displayed:



The password is...

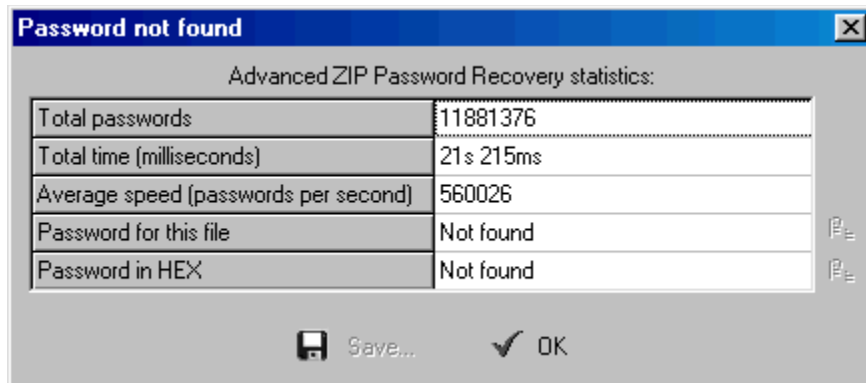
When (if) the password is found, the program shows it, as well as the number of passwords which have been tested, and the program speed:



The last line displays the password in HEX form – it might be useful if the password, for example, contains some non-English characters which cannot be displayed correctly on your system (due to missing fonts, etc.).

Pressing the small button at the right of the password (in "normal" or HEX form) copy the password into the Clipboard. Alternatively, you can save the password to a file.

If all possible passwords in the given range have been tried without success (without finding a valid one), the message looks like:



If you stopped recovery by pressing the "Stop" button, the current step of the brute-force attack is saved in the "Start from" field. Now you can press the "Start" button again. Recovery will be continued from this step.

Known bugs and limitations

- When the files in the archive are "stored" (without compression, only with encryption), the performance might be lower than expected (especially on large files), because decrypting the whole file is required (unless you know the first few bytes of the encrypted file).
- If the archive contains two or more encrypted files, the program assumes that all of them are encrypted with the same password. See the next help topic: [Files with different passwords](#).
- ZIP archives with files compressed using PKWARE Data Compression Library Imploding (dclimplode) method are not supported (will be supported if we'll get at least a few requests).
- The program may fail to recover the password for file compressed with **deflate64** method (available in PKZip 4.0). We haven't found any problems here so far, but the chance is out there. That's just because PKWARE doesn't provide proper descriptions of this method. When the attach is started and this method is detected, the program displays an appropriate warning message.

Files with different passwords

If the files inside the archive have been encrypted with different passwords, **AZPR** might not be able to find the correct password. The workaround is to (1) make a backup copy of your archive; (2) remove all files from the archive except those which definitely have the same password (perhaps just one file); and (3) run **AZPR** on the resulting archive. When (if) **AZPR** finds the correct password, create another new archive, keeping the next portion of files which have the same password. If **all** the files have different passwords, you're in trouble – too much time for recovering them will be required; but that's the only thing you can do.

What to start from

If you have no idea how long the password is and what characters it may contain, just run the dictionary-based attack first. If it fails, try brute-force with the following options (character set and password length range):

<u>Charset</u>	<u>Length</u>	<u>Passwords</u>	<u>Time</u>
All printable	1..6	742,912,032,768	14 hours
Digits, small/capital, space	7	3,938,980,724,736	73 hours
Digits, small letters, space	8	3,512,479,514,624	65 hours
Digits, capital letters, space	8	3,512,479,514,624	65 hours
Digits	9..12	3,452,035,661,824	64 hours

The third column shows the total number of possible password combinations (with the given charset and password length); and the last column shows the maximum time (approximated) required for recovering the password, assuming that the speed is 15,000,000 passwords per second (the performance of a computer with Pentium III CPU for ZIP archives with five or more encrypted files).

Dictionary-based attacks

As noted above, dictionary-based attack is *very* effective, so please try it first. Moreover, if you know the "structure" of the password (for example, the characters at some positions), it is recommended that you create your own dictionary based on the rules you have. There are a lot of dictionary generators around; some (developed by third parties) are available from our **Password Recovery Software** page:

<http://www.elcomsoft.com/prs.html>

The password generator may also help if you "almost" remember the password. You might have missed one or two characters, or typed an extra one, or just missed a few characters – some generators allow "mutating" the password and can print/save all similar ones (creating a new wordlist or dictionary, which can be used with **AZPR**).

Command line

You can run **AZPR** with command-line parameters. The syntax is:

AZPR [switches] [zip-filename]

or

AZPR [switches] [azr-filename]

The switches are separated with / or - characters. If the switch is followed by some data (e.g., filename, starting password, etc.) which contains these characters: space, semicolon, slash or dash, it must be enclosed in (single or double) quotes.

<u>Switch</u>	<u>Description</u>	<u>Default</u>
/a:b m d p	attack type (brute-force, mask, dictionary, plaintext)	brute-force
/c:csdepa	character set (caps, small, digits, special, space, all)	caps
/u:chars	user-defined charset	
/oem	convert to OEM (for user-defined charset and dictionary attack)	disabled
/sf:pass	start from password	
/usewz:X	use optimized WinZip attack	
/useknownstart:XX	use known bytes in stored file (from 1 to 4 hex values, no spaces)	
/m:mask	mask	
/ms:C	mask symbol	?
/min:N	minimum password length	1
/max:N	maximum password length	5
/usewz:X	use optimized WinZip attack	
/useknownstart:XX	use known bytes in stored file (from 1 to 4 hex values, no spaces)	
/d[:filename]	dictionary filename	
/sm	smart mutations	disabled
/ac	try all possible upper/lower case combinations	disabled
/sl:N	start from line N	0
/p[:filename]	plaintext filename	
/autosave:N	autosave every N minutes; 0 means disabled	5
/aname:filename	autosave filename	
/adir:dir	autosave directory	
/idle	run at idle priority	enabled
/high	run at high priority	disabled
/dontstart	don't start the attack, just load/set the parameters	
/minimize	Minimize the program after starting the attack	
/smartexit[:filename]	When the attack is completed, write all statistics, including the password (if found) to the given file (default "cmdline_stats.txt"), and close the program.	disabled

Examples:

azpr.exe /a:b /c:cs /min:3 /max:7 /smartexit test.zip
(brute-force attack; small and capital letters; length from 3 to 7; save and exit when done)

azpr.exe /a:b /u:12345abcde test.zip
(brute-force attack with "12345abcde" character set; length: from 1 to 5)

azpr.exe /a:m /c:d /m:june???? /sf:june1000 /high test.zip
(mask attack with "june????" mask; charset: digits; high priority)

azpr.exe /d:english.dic /sm /oem /dontstart test.zip
(dictionary attack; dictionary: "english.dic"; smart mutations; convert words from ANSI to OEM; don't start)

azpr.exe /a:p /p:plain.zip test.zip
(known plaintext attack)

If the parameter is the azr-file, the program will immediately load all the settings from it (ignoring the other settings supplied in the command line, except **/dontstart**, **/minimize** and **/smartexit**), and run the attack.

Unexpected crash

We've tried to make **AZPR** as reliable as possible, but on some specific archives it may still crash (after a few hours or even days of continuous operation). However, if you have the Auto-save option enabled, it is safe – the program will restart automatically (from the last-saved point). In addition, it will create a special dump file (in the same folder where **azpr.exe** is located), in the form:

CrashLog XXX, YYY.log

where XXX indicates the crash date, and YYY the crash time. The file contains information which will really help us to fix the problem (the crash address, registers map, stack dump, process list, memory map, etc.) – so, if you should ever get that dump (an appropriate message will be written into the message log), please send it to us, and we'll do our best to make **AZPR** even better.

To test the functionality described above, press **Ctrl-Alt-Shift-F12** – the program will simulate an exception by causing a division by zero.

Contacting us

For technical support, please contact us at support@elcomsoft.com. In the subject of your mail, please write "AZPR x.y" (where x.y is the version number), followed by "problem", "suggestion" or whatever else.

Where to get the latest version

The latest version of **AZPR** is always available from our web page at <http://www.elcomsoft.com/azpr.html>. Other password recovery products (for Microsoft Office – Word, Excel, Access, PowerPoint, Visio, Outlook, VBA; MS Backup, MS Project; MS Mail, MS Schedule+ Lotus Organizer, Lotus 1-2-3, Lotus WordPro, Lotus Approach; Borland/Corel Paradox, Symantec ACT!, Intuit Quicken and QuickBooks, Adobe Acrobat PDF) are available from our server at <http://www.elcomsoft.com/prs.html>.

If you'd like to receive notifications about new releases of **AZPR**, please subscribe to our mailing list; mailing list archive and subscription information is available at <http://prsoftware.listbot.com>.

Copyright and license

- All copyrights to **AZPR** are exclusively owned by ElcomSoft Co.Ltd.
- Anyone may use this software during a test period of 30 days. Following this test period of 30 days or less, if you wish to continue to use **AZPR**, you MUST register.
- Once registered, the user is granted a non-exclusive license to use **AZPR** on one computer (i.e. a single CPU), for any legal purpose, at a time. The registered **AZPR** software may not be rented or leased but may be permanently transferred, if the person receiving it agrees to the terms of this license. If the software is an update, the transfer must include the update and all previous versions.
- The **AZPR** unregistered (trial) version may be freely distributed provided the distribution package is not modified. No person or company may charge a fee for the distribution of **AZPR** without written permission from the copyright holder.
- **AZPR** IS DISTRIBUTED "AS IS". NO WARRANTY OF ANY KIND IS EXPRESSED OR IMPLIED. YOU USE IT AT YOUR OWN RISK. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.
- You may not use, copy, emulate, clone, rent, lease, sell, modify, decompile, disassemble, otherwise reverse engineer, or transfer the licensed program, or any subset of the licensed program, except as provided for in this agreement. Any such unauthorized use shall result in immediate and automatic termination of this license and may result in criminal and/or civil prosecution.

All rights not expressly granted here are reserved by ElcomSoft Co.Ltd.

- Installing and using **AZPR** signifies acceptance of these terms and conditions of the license.
- If you do not agree with the terms of this license you must remove **AZPR** files from your storage devices and cease to use the product.

Registration

You can register your copy of **AZPR** at a cost of \$60 (personal license) or \$120 (business license). With the personal license, you can use the program for non-commercial purposes in a non-business, non-commercial environment. To use the program in a corporate, government or business environment, you should purchase a commercial license. Select one of the following ways:

- Through the World Wide Web. This is the fastest and easiest way. Your credit card information is sent directly to the credit card processor in a secure manner, so that nobody else can see it. This protects you by ensuring that nobody but you and the credit card processor will see your card.

For details on ordering **AZPR** through the Web, please go to the **AZPR** page at:

<http://www.elcomsoft.com/azpr.html>

and follow the link "Order the Fully Licensed Version of **AZPR**". Or, you can go directly to the order page:

<https://www.regnow.com/softsell/nph-softsell.cgi?item=1170-6>

- You can also use the wire transfer (to our bank account), or purchase order, or send us a personal or company check – look at the order form (*order.frm*). Phone and fax orders are accepted as well. Please don't send cash to our mailing address.

On payment approval (usually within a few minutes after you register online), we'll send you the registration key which will remove all limitations of the unregistered version, such as maximum password length, etc. Your registration will be valid for **all** future versions of **AZPR**. To enter the registration code into the program, simply press the **Register** button on [Other options](#) page.

If you do not receive your registration key within a reasonable amount of time (two business days for credit card payments or two weeks for other payments), please notify us about that! We're very sorry for any inconvenience caused by those delays.

Limitations of unregistered version

An unregistered version of **AZPR** has the following limitations:

- with a brute-force attack, passwords longer than 5 characters cannot be recovered
- some dictionary attack options are disabled
- in known-plaintext attack:
 - the decryption keys are not shown, and only the first file from an archive can be decrypted
 - attack cannot be resumed

Acknowledgements

Many people have helped make **AZPR** what it is, by making suggestions, helping test, reporting bugs, etc., but particular thanks goes to the following individuals: **Irina Katalova**, Alexander Katalov Jr, Dmitry Sklyarov, Alexander Volok, Marco D'Amato, John Taylor, Paolo Viappiani, Darren Parker. And the special thanks to **AI Anway** for correcting the documentation.

This product includes cryptographic software written by Eric Young.

