

NAME

sa5 – SecretAgent® file encryption, digital signature, and certificate management utility

CONTENTS

Synopsis	Data Recovery
Description	Diagnostics
Availability	See Also
Options	Authors
Examples	Acknowledgements
Caveats	Technical Support
Files	Legal Information

SYNOPSIS

View command line syntax

sa5 [-h]

Generate a new key pair

sa5 -g userID -t type -o file [-f] [{ -Q | [-S] -p pwd }] [{ -q | -i [2] }] [-r seed] [-w level] [-D digest] [-F encoding]

Import a PKCS#12 file

sa5 -g userID [{ -Q | [-S] -p pwd }] [{ -q | -i [2] }] [-f] [-w level] p12file

Encrypt (and optionally sign)

sa5 -e recipients [-f] [-l] [-n] [-o {file/path}] [-s signer [, CAPI]] [{ -Q | [-S] -p pwd }] [{ -q | -i [2] }] [-r seed] [-w level] [-z] [-A] [-C compression_alg] [-E cipher] [-F encoding] [-L format] [-Z filter_type] [@file / file... / dir...]

Sign (without encrypting)

sa5 -e . -s signer [, CAPI] [-f] [-o {file/path}] [{ -q | -i [2] }] [-r seed] [{ -Q | [-S] -p pwd }] [-w level] [-A] [-l] [-n] [-z] [-F encoding] [-L format] [-N] [-Z filter_type] [@file / file... / dir...]

Decrypt (and verify any signatures)

sa5 -d {userID | khpath | CAPI} [{ -Q | [-S] -p pwd }] [-f] [-o path] [{ -q | -i [2] }] [-r seed] [-s cert] [-w level] [-z] [-A] [-Z filter_type] [file... / dir...]

Verify a signed-only file

sa5 -d . -s cert [{ -q | -i [2] }] [-w level] [-z] [-V msg_file] [-A] [-Z filter_type] [file... / dir...]

Inspect encrypted and/or signed archives

sa5 [-w level] file...

Inspect (and optionally validate) a certificate

sa5 [-s ca_cert] [-I CApth [-R CRLpath]] [-w level] subject_cert

List the labels for all certificate slots on a PKCS#11 token

sa5 -w level library

Save to files all certificates associated with a particular slot on a PKCS#11 token

sa5 [{ **-q** | **-i** [2] }] [**-o** path] **-w** level label,library

Export a PKCS#12 file

sa5 [{ **-q** | **-i** [2] }] **-x** p12file **-s** userID **-p** pwd [**-w** level] file ...

Export a PKCS#7 file

sa5 [{ **-q** | **-i** [2] }] **-x** p7bfile [**-w** level] file ...

Change a PKCS#8 password

sa5 **-y** { **-Q** | [**-S**] **-c** new_pwd } [{ **-Q** | [**-S**] **-p** pwd }] [**-f**] [**-w** level]
prvfile

Zap

sa5 **-z** [**-w** level] [@file / file ... / dir ...]

Calculate a message digest

sa5 **-H** [**-D** digest] [**-w** level] file

Index CA certificates

sa5 **-I** CApath [**-w** level]

Index CRLs

sa5 **-R** CRLpath [**-w** level]

Reseed the PRNG

sa5 **-r** seed

Display the active security policy settings

sa5 **-O**

Suggest a NIST FIPS 181 password

sa5 **-P** length

DESCRIPTION

sa5 is an X.509 certificate-based file encryption and digital signature utility, which also provides some credential management functions. It may be used to encrypt one or more files into a single ciphertext archive to facilitate secure local storage or for transmission to one or more recipients.

Ciphertext archives may be created in the native SecretAgent *.sa5* archive format, or as S/MIME CMS PDUs with or without MIME headers (see RFC 3369). On Windows, the SecretAgent 5 GUI also supports OpenPGP (RFC 2440) as an optional input and output format. All versions of the product support the optional compression of plaintext prior to its encryption. Several types of output files may be produced in raw binary format, or base64-encoded to produce printable ASCII text.

sa5 may also be used to generate RSA, DSA/DH or ECC key pairs, and either self-signed certificates or PKCS#10 certificate requests. Among the cryptographic operations supported by **sa5** (with their corresponding principal command line options) are the following:

generate keypair	-g
parse PKCS#12 file (import)	-g
change private key password	-c
encrypt and/or sign	-e
decrypt and/or validate	-d

create certificate renewal request	-x
export p12 credentials or p7 cert chain	-x
add entropy to random number generator	-r
zap (i.e., securely erase) a file	-z
hash a file	-H
index a CA certificate directory	-I
index a CRL certificate directory	-R
list slot labels on a PKCS#11 token	-w
extract certificate from a PKCS#11 token	-w
suggest a pronounceable password	-P
inspect cipher archive or certificate	<i>file</i>

In the command line syntax summary, square braces ([]) denote optional arguments. When two or more arguments appear in curly braces ({}), exactly one must be chosen. When options without arguments are grouped together behind a single dash, only the last option in the group can take an argument. For example, **-H -D3** is equivalent to **-HD3**. Option arguments containing spaces must be double-quoted. When a comma-delimited argument list is provided for a single option, the entire list must be wrapped in a single pair of double quotes if any element in the list contains a space.

Under Windows, the SecretAgent command line is named **sa5cli** rather than **sa5**, as the latter filename has historically been used for the GUI version.

AVAILABILITY

sa5 is highly portable and interoperable between all supported platforms. It is currently available for Windows and all popular UNIX systems. Currently supported platforms include:

Windows 9x/NT4/2000/ME/XP/2003	Solaris 7-10 (SPARC and x86)
PocketPC 2002/ARM (WinCE)	Mac OS X (PPC)
PocketPC 2003 (Windows Mobile)	Compaq Tru64 (Alpha)
Linux (Intel x86)	OpenVMS/AXP
HP-UX 11.x (PA-RISC)	SGI IRIX 6.x (MIPS)
IBM AIX 4.x (PPC)	Cray UNICOS (C90/T90/T3E)

OPTIONS

The following options are supported. Argument values in bold denote default values.

-@ *cfg_file*

Process command line arguments in *cfg_file*. Multiple **-@** arguments are processed in the order in which they appear on the command line. (If it exists in the current directory, the special configuration file *sa5.cfg* is processed automatically by **sa5** before it processes its command line. Therefore, specifying **-@sa5.cfg** on the command line will cause the file to be processed twice.)

-a *sar_files*

Use the comma-delimited list of *.sar* files (containing key recovery shares) to complete the data recovery process, i.e., to decrypt a particular *.sa5*, *.saa*, or *.exe* file.

-d { *userID* | *khpath* | **CAPI** | *label,library* | . }

Decrypt and/or validate file arguments. For decryption, either the single private key file *userID.prv* is used, or all private key files (assumed to have the same password) in the specified private key history directory *khpath* are tried.

Under Windows, you can attempt to decrypt with the private keys in your Microsoft CAPI store (or on a hardware token under control of CAPI) by specifying the string literal **CAPI**.

To decrypt using a key pair on a PKCS#11 token, specify the *label* of the appropriate certificate slot and PKCS#11 *library* name. (To list the slot labels for a particular token, use the PKCS#11 inspect commands.)

To validate a non-encrypted, signed archive, use a **-d** argument consisting of a single period.

- e** { *recipients* | . }
 Encrypt and/or sign all file arguments. When encrypting, *recipients* consists of a comma-delimited list of certificate and PKCS#7 files. The filename extensions *.cer*, *.der* and *.crt* may be omitted for individual certificate files; but *.p7b* or *.p7c* must be specific for PKCS#7 files. For sign-only operations, the **-e** argument must consist of a single period. When creating a password-protected, self-decrypting executable, the **-e** option can be used to specify KRA certificates; otherwise, its argument should be a single period.
- f** Suppress display of program name and copyright notice.
- g** *userID*
 Generate a keypair (consisting of a private key and either a self-signed certificate or a PKCS#10 certificate request) with common name *userID*. Use **-t** to specify the desired type of key, **-o** the output filename and format. **-g** may also be used to "import" a PKCS#12 file, in which case only the p12 password (that is also used to protect the output *.prv* file) is required.
- h** Display command line syntax summary and exit.
- i** Interactive mode. Prompt before overwriting. Specifying **-i2** causes the program to terminate with an error code if the user, when prompted to overwrite a file, chooses not to do so.
- k** Overwrite READONLY files during autoencryption/decryption.
- l** Follow symbolic links. Applies when recursing directories with **-A** or otherwise enumerating files.
- n** Disable keyUsage and validity checks (with **-e**). Does not apply when generating CMS signatures.
- o** { *file* | *path* }
 Use the specified output *file* for key generation, encryption, signing, and certificate extraction (during signature validation) operations. Use specified output *path* to redirect all output when decrypting, etc.
- If an output *file* is specified when generating a new key pair with the **-g** option, it must have one of the following extensions:
- | | |
|-------------|--|
| <i>.crq</i> | produce a PKCS#10 certificate request file |
| <i>.cer</i> | produce a self-signed certificate |
- If an output *file* is specified when encrypting and/or signing, it must have one of the following extensions (which is used to determine the type and format of the output file to be produced):
- | | |
|-------------|--|
| <i>.sa5</i> | encrypt all input files into a single archive using the cipher specified by the -E option; defaults to AES-128-CBC (-E12) |
| <i>.saa</i> | encrypt "in place" with AES-128-CBC (-E12); any other cipher specification is illegal in this context and is ignored |
| <i>.sgn</i> | sign, creating the specified detached signature file |
| <i>.exe</i> | encrypt to a self-extracting executable using RC4; produces a Windows <i>.exe</i> file, unless -X is used to specify an alternate target system |
| <i>.p7m</i> | encrypt and/or sign to a CMS PDU (implies -L1) |
- p** *pwd*
 Use the password *pwd* for all private key operations, and as the PBE password when creating a self-decrypting archive. (Causes the display to be cleared during command line processing when preceded by **-S**.)
- q** Quiet mode. Overwrite existing files without interactively prompting for permission.
- r** *seed* Use supplied *seed* as an input for additional entropy when seeding the internal pseudo-random number generator. (This option is largely unnecessary as the program automatically uses extensive platform-dependent system state information to seed its internal PRNG upon start-up.)

-s { *signer*[,**CAPI**] | *cert*[,*label*,*library*] | *ca_cert* }

When signing, use *signer.prv* as the signer's private key file. (In this case, the **-p** argument must supply the file's password or the user will be prompted to enter it at runtime.)

Under Windows, you can sign with a private key in a Microsoft CAPI store (or on a hardware token under control of CAPI) by appending the string literal **,CAPI** to the name of your certificate file (without its *.cer*, *.der* or *.crt* extension).

To sign with a key pair on a PKCS#11 token, append the *label* of the corresponding certificate slot and PKCS#11 *library* name to the name of your certificate file (without its *.cer*, *.der* or *.crt* extension). (To list the slot labels for a particular token, or to copy a certificate from the token to a certificate file, use the PKCS#11 **-w inspect** commands.)

When validating a detached or enveloped signature, if *cert.cer* doesn't already exist, it is used to capture a copy of the signer's certificate. If *cert.cer* does exist, it is regarded as the issuing CA's certificate and used to validate the signer's certificate.

When validating a certificate, *ca_cert.cer* is regarded as the issuing CA's certificate. For complete path validation, use **-I** to provide a (previously indexed) directory containing CA certificates. If CRL checking is desired, use **-R** to provide a (previously indexed) directory of CRLs. (Certificate and CRL directories must be indexed before use for path validation. See **-I** and **-R** options for details.)

-t type Generate a key pair of the indicated *type*, i.e., load the parameter file *type.kyp* and use it for key generation. For a list of the supplied RSA, DSA/DH, and ECC parameter files (**.kyp*), see the FILES section below. The current default *type* is **dsa-1024**.

-x {*p12file* | *p7bfile*}

Export appropriate components of the supplied keypair (with an optional certificate chain) as a PKCS#12 or PKCS#7 PDU with the specified filename. PKCS#12 files require the owner's private key file and password.

-w level

Set output message *level* to one of the following decimal digits:

- 0** no output [default]
- 1** terse progress and diagnostic messages
- 2** more detailed progress and diagnostic messages
- 3** when used with **-d**, displays the filenames in an encrypted archive without actually producing plaintext output

-z Zap all input files upon successful completion of current encrypt/decrypt operation. Securely erase all file arguments in a manner compliant with the new DoD object reuse guidelines. (See *Department of Defense Magnetic Remanance Security Guideline* CSC-STD 005-85). Here, an operation is considered to have completed successfully if all input files were successfully processed, and either the **-q** option was in effect or the user chose to overwrite all files. If **-q** was not specified and the user skipped the overwriting of even one output file, the operation is considered to have failed and input files are not erased.

-A Recurse directories (for encryption, decrypting, signing, validating, and zapping).

-C compression_alg

Compress plaintext prior to encryption using the compression algorithm encoded by *compression_alg*:

- 0** no compression [default]
- 1** LZSS (a Lempel-Ziv variant)

-D digest

Use the message digest (hash function) encoded by *digest* :

- 1** MD2 (RFC 1319)
- 3** MD5 (RFC 1321)
- 4** SHA-1 (FIPS 180-1, ANSI X9.30(2)) [default]
- 5** SHA-256

6	SHA-384
7	SHA-512
8	SHA-224

when hashing files with the **-H** option. When creating self-signed certificates and PKCS#10 certificate request files (RFC 2314), only the *digest* values 1, 3, and 4 are currently supported.

-E *cipher*

Use the specified symmetric *cipher* for encryption:

2	DES-CBC (FIPS 46-2/FIPS 81/ANSI X3.92/X3.106) [deprecated]
4	TDES-CBC (FIPS 46-3/ANSI X9.52)
6	EA2-CBC (AT&T proprietary cipher) [deprecated]
8	DESX-CBC (draft-simpson-desx-02.txt)
12	AES-128-CBC (FIPS 197) [default]
14	AES-192-CBC (FIPS 197)
16	AES-256-CBC (FIPS 197)

With earlier releases of **sa5** one could select ECB modes of the above ciphers using odd *cipher* values, but this is no longer supported for encryption. Archives encrypted using an ECB mode can still be decrypted, however. Contact ISC for details.

-F *encoding*

Apply the following *encoding* to output files:

0	no encoding (binary output) [default]
1	base64 encoding (ASCII output)

-H Hash file arguments. Use **-D** to specify a message digest function if something other than the default SHA-1 algorithm is desired.

-I *CApath*

Specifies the search directory for root and intermediate CA certificates when certificate path validation is performed. When used alone, creates or updates the certificate index file in the specified directory.

-L *format*

Produce output files in the specified *format*:

0	SecretAgent 5 (.sa5) [default]
1	CMS/SMIME (with internal MIME headers; implies -N)
2	CMS/RAW (without MIME headers)

-N Create an enveloped (as opposed to a detached) signature when used with **-s**. Implied by **-L1**.

-R *CRLpath*

Specifies the search directory for CRLs when certificate path validation is performed (for digital signature or certificate validation). When used alone, creates or updates the CRL index file in the specified directory.

-S Clear display in order to hide passwords typed on the command line. (The switch **-S** must appear before the **-p** option or it will not function correctly.)

-Q Use a NULL password. When specified, the user is not prompted to enter a password for any operation; an empty password is provided automatically.

-V *msg_file*

Validate a detached CMS signature against the message body contained in *msg_file*. Must be used when validating detached CMS signatures.

-X *stub*

Create a self-decrypting, RC4-encrypted archive based on the executable *stub.exe*, whose file-name identifies the intended target system. The current list of available *stub* names with their respective targets is:

stbaixppcrc4	IBM AIX/PPC (RC4)
stbhxparrc4	HP-UX/PA-RISC (RC4)
stbirxmiprc4	SGI IRIX/MIPS (RC4)

stblnxx86rc4	Linux/i86 (RC4)
stbosxpprc4	MAC OS X/PPC (RC4)
stbsolsparc4	Solaris/SPARC (RC4)
stbsolx86rc4	Solaris/x86 (RC4)
stbtrualprc4	Compaq Tru64/Alpha (RC4)
stbwinx86rc4	Windows/x86 (RC4) [default]

-Z *filter_type*

This switch allows the program to be used as a UNIX-like filter, possibly as part of a pipeline of commands. Works on all platforms, but only with *.sa5* archives. Permissible values of *filter_type* are the following:

0	input from file, output to file [default]
1	input from <code>stdin</code> , output to file
2	input from file, output to <code>stdout</code>
3	input from <code>stdin</code> , output to <code>stdout</code>

Since compression is automatically enabled (and cannot be disabled) when **-Z1** and **-Z3** are used to encrypt from `stdin`, these options should probably not be used with a large volume of incompressible data.

EXAMPLES

This section contains detailed examples illustrating the use of each of the most popular **sa5** functions.

Generating a key pair

The command:

```
sa5 -w1 -g"Bill Clinton" -p testtest -t rsa-1024 -o bill.crq
```

will create a 1024-bit RSA keypair and output the public key as a (signed) PKCS#10 certificate request, *bill.crq*, and the private key as a PKCS#8 file, *bill.prv*. Similarly, the command:

```
sa5 -w1 -g"Bill Clinton" -p testtest -t rsa-1024 -o bill.cer
```

will create a self-signed certificate file *bill.cer* and a private key file *bill.prv*.

NOTE: To specify X.509 DN components for the certificate subject, see Configuration Files below.

Encrypting

WARNING: Unlike the SecretAgent 5 GUIs for UNIX and Windows, the **sa5** command line does not automatically validate each recipient's certificate prior to its use for encryption. Certificate validation, if desired, must be performed separately. Details are provided below.

The command:

```
sa5 -e bob -o archive.sa5 plain1.txt plain2.txt
```

will use the certificate *bob.cer* to encrypt the files *plain1.txt* and *plain2.txt* to produce the single (binary) ciphertext output file *archive.sa5*.

Note that the explicit specification of the *.sa5* extension to the **-o** argument tells **sa5** to produce a standard *.sa5* archive. **-F1** may be added to the command line to produce a base64-encoded (ASCII) ciphertext archive.

More than one recipient is specified by providing a comma-delimited list of certificate filenames as the **-e** argument:

```
sa5 -e ann,bob -o archive.sa5 plain1.txt plain2.txt
```

Now, either Ann or Bob can decrypt the ciphertext archive *archive.sa5*.

To produce individual *.sa5* ciphertext archives for each input file, simply omit the **-o** option:

```
sa5 -e ann,bob plain1.txt plain2.txt
```

This produces *plain1.sa5* and *plain2.sa5*, both encrypted with *ann.cer* and *bob.cer*.

The encryption command may be used as a filter, with the plaintext to be encrypted being piped in via `stdin`. For example:

```
type testdata.txt | sa5 -e test -Z1 testdata.txt
```

will output the ciphertext archive `testdata.txt.sa5`. Note that, even though the plaintext is taken from `stdin`, a filename argument is still be specified so that program can construct an output filename.

This command is simliar to the previous one, but with base64-encoded ciphertext directed to `stdout`:

```
type testdata.txt | sa5 -F1 -e test -Z3 testdata.txt
```

This command encrypts the plaintext in an input file `testdata.txt` and produces base64-encoded ciphertext output on `stdout`:

```
sa5 -F1 -e test -Z2 testdata.txt
```

To implement a shared secret, data recovery scheme, several (non-RSA) certificate files of exactly the *same key size and type* can be combined with a plus sign (+) and treated as a single virtual recipient in the `-e` argument list:

```
sa5 -e ann+bob,eve plain1.txt plain2.txt
```

For further information, see the DATA RECOVERY section below.

This command produces a self-decrypting Windows executable (i.e., a `.exe` file) using the password `pwd`:

```
sa5 -e . -p pwd -o archive.exe plain1.txt plain2.txt
```

Again, the output ciphertext file format is selected by explicitly specifying the `.exe` extension on the output filename provided as the `-o` argument. If you omit the `-p` argument, you will be interactively prompted for the password at runtime. Use `-X` to specify a target platform other than the Windows default.

To support key recovery with a self-decrypting archive, specify the list of KRA certificate filenames as the `-e` argument in place of the period:

```
sa5 -e kra1,kra2 -p pwd -o archive.exe plain1.txt plain2.txt
```

for two individual KRAs `kra1` and `kra2`, or

```
sa5 -e kra1+kra2 -p pwd -o archive.exe plain1.txt plain2.txt
```

to force the two KRAs to cooperate (in a secret sharing scheme) in order to perform the data recovery process. (See the DATA RECOVERY section below.)

Decrypting (and verifying)

To decrypt `file.sa5` using the password-protected private key `ann.prv` with password `annspwd`:

```
sa5 -d ann -p annspwd file.sa5
```

The same command with all plaintext output redirected to the directory `'/tmp/plaintext'`:

```
sa5 -d ann -p annspwd -o /tmp/plaintext file.sa5
```

If `file.sa5` has also been signed, you will be notified of the validity of its signature. In this case, you may use the `-s` option to either specify a CA certificate against which the purported signer's certificate is to be validated, or to specify a file into which the signer's certificate is to be saved:

```
sa5 -d ann -p annspwd -s user file.sa5
```

In this case, if `user.cer` already exists, it is assumed to contain the certificate of the CA who issued the certificate of the purported signer of `file.sa5`, and it is used to validate that certificate. If `user.cer` doesn't already exist, the signer's certificate is extracted from `file.sa5` during the decryption/validation process and written into it.

Signing without encrypting

To sign *without* encrypting, specify an empty recipient list with a period (`-e .`) and provide your private key and password as in the following example:

```
sa5 -e . -s ann -p annspwd message.xyz
```

If you do not provide your password on the command line, you will be interactively prompted to enter it during execution.

Verifying an (unencrypted) signed file

As long as it is not also encrypted, you may verify the signature on a signed archive using a command of the form:

```
sa5 -d . -s user file.sa5
```

If *user.cer* already exists, it is assumed to contain the certificate of the CA who issued the certificate of the purported signer of *file.sa5*, and it is used to validate that certificate. Otherwise, the signer's certificate is extracted from *file.sa5* during the validation process and saved in *user.cer*.

NOTE: To validate the signature on an encrypted file, you must provide a private key and password to decrypt it. Since it is always plaintext that is signed, not ciphertext, you must allow **sa5** to recover the plaintext in order to validate the signature on it.

Inspecting an encrypted archive

The command:

```
sa5 file.sa5
```

will report on the format and cipher/mode used to create *file.sa5* as well as display the originator (if the archive is signed) and all recipient DNs. **sa5** can inspect only one file per invocation.

Erasing a sensitive file

To zap (i.e., securely erase) one or more files, use a command of the form:

```
sa5 -z file1 file2 ...
```

The specified files will be overwritten according to the current DoD Magnetic Remanance Guidelines and then unlinked.

WARNING: This operation is non-interactive; the specified files will be immediately and irretrievably wiped from your hard disk.

Inspecting and/or validating a certificate

The command:

```
sa5 user.cer
```

will display the subject and issuer distinguished names (DNs) and validity period of the X.509 certificate in *user.cer*

To validate a subject certificate against the certificate of its issuing CA, use the `-s` option to specify the name of the CA's certificate file, in this case *issuer.cer*:

```
sa5 -s issuer subject.cer
```

An entire certificate path may be validated by separately invoking **sa5** to validate each successive link in the path, but there is a faster and easier way to do it. Simply place all CA certificate files in a dedicated directory, say */var/cacerts*, and create an *index* file that directory using the command:

```
sa5 -I /var/cacerts
```

Assuming that all required CA certificates are in the directory */var/cacerts*, and that that directory has been previously indexed, the command:

```
sa5 -I /var/cacerts subject.cer
```

will attempt to automatically discover, and validate, a certificate path linking *subject.cer* back to a (self-

signed) root certificate in */var/cacerts*.

If CRL checking is desired, place all CRL files in a dedicated directory (which must be separate from the CA certificate directory), say */var/crls*, and index it using the command:

```
sa5 -R /var/crls
```

Then the command:

```
sa5 -I /var/cacerts -R /var/crls subject.cer
```

will automatically perform full certificate path validation with CRL checking.

NOTE: When the **-I** and **-R** options are specified, and CRL checking has been enabled via the *sa5so.cfg* file, valid CRLs for *every* CA in a given certificate path must be found in the specified **-I** directory, and both directories must be indexed, or validation of the path will fail.

Importing a PKCS#12 file

The command:

```
sa5 -g ann -p AnnsPwd AnnsP12.p12
```

will "import" the PKCS#12 file *AnnsP12.p12* with password *AnnsPwd*, splitting it into a (PBE-encrypted PKCS#8) private key file *ann.prv* and a certificate file *ann.cer*. If *AnnsP12.p12* also contains a chain of CA certificates, they will be written to successively numbered files: *ann1.cer*, *ann2.cer*, etc., with the last file in the series being that of the self-signed root certificate terminating Ann's certificate path.

Exporting a PKCS#12 file

The command:

```
sa5 -x ann.p12 -s ~/certs/ann -p AnnsPwd
```

will use *AnnsPwd* to extract Ann's private key from the existing PBE-encrypted PKCS#8 file *~/certs/ann.prv* and combine it with her existing certificate in *~/certs/ann.cer*, to create the PKCS#12 *ann.p12* (which is also encrypted with *AnnsPwd* and output to the current directory).

To include additional CA certificates in the output PKCS#12 file, simply append them to the command line. For example, the command:

```
sa5 -x ann.p12 -s ~/certs/ann -p AnnsPwd ~/certs/ca1.cer ~/certs/rootca.cer
```

includes the two CA certificates in *ann.p12* along with Ann's personal certificate.

Exporting a PKCS#7 file

The command:

```
sa5 -x ann.p7b ~/certs/ann.cer ~/certs/ca1.cer ~/certs/rootca.cer
```

creates a PKCS#7 file, *ann.p7b*, containing the three specified certificates (which presumably constitute Ann's entire certificate path). Ann may give the file *ann.p7b* to anyone who wishes to encrypt files for her, or to validate her signatures with full certificate path validation.

Hashing a file

The command:

```
sa5 -w1 -H -D7 message.txt
```

displays the SHA-512 message digest (**-D7**) of the file *message.txt*. If the **-D** option is omitted, the file's SHA-1 message digest is produced.

CAVEATS

Simple filenames specified on the command line are assumed to be in the current directory. Use full or relative pathnames to access files in other locations.

When providing a list of arguments, at least one element of which contains a space, use only one set of quotes and place them around the entire argument list. For example:

```
sa5 -w 2 -e "C:\Program Files\SecretAgent 5\ann,c:\bob" file.txt
```

will encrypt *file.txt* using the certificate files *C:\Program Files\SecretAgent 5\ann.cer* and *c:\bob.cer*. Note that the command

```
sa5 -w2 -R -e"long path\ann,long path\bob" file1.txt
```

is correct, but

```
sa5 -w2 -R -e"long path\ann","long path\bob" file1.txt
```

is not.

You may wish to set the **SA5HOME** environment variable (**RANDDIR** on Windows) to the directory containing the **sa5** executable and its support files. Once this is done, simply adding the install directory to your **PATH** should allow you to successfully run **sa5** regardless of your current working directory. If the appropriate environment variable is not set, **sa5** will fail to run if its support files are not in the current working directory.

When using **.cer* or **.prv* files as command line arguments, **sa5** automatically appends the proper extension to the specified filename. Consequently, filename extension are generally not explicitly specified on the **sa5** command line. The only exceptions to this rule are the following:

- @ takes the complete pathname of its configuration file argument,
- o takes a complete output filename whose extension is used to determine the type/format of output, and
- s requires a complete filename when performing certificate path validation

When encrypting, the output filename extension (*.sa5*, *.saa*, or *.exe*) specified on the command line is used to determine the format of the ciphertext archive. If *.exe* is specified without an explicit **-X** argument, **sa5** will use the supplied *self32.exe* stub to create an RC4-encrypted self-decrypting **Windows** executable (with a key of at most 320 bits based on the supplied password).

When using the **-Z** option to make **sa5** act as a filter, the following restrictions are imposed:

- *.exe* files cannot be decrypted from *stdin*, though they can be decrypted to *stdout*
- *.saa* files cannot be encrypted to *stdout*, nor decrypted from *stdin*
- only *.sa5*-based file formats are supported; CMS format is not supported
- key recovery operations are not supported as a filter

Some file systems do not support file sizes greater than or equal to 2GB (2^{31} bytes). Use such files with caution, especially when cross-platform interoperability is desired.

You may wish to set the **SA5HOME** environment variable (**RANDDIR** on Windows) to the directory containing the **sa5** executable and its support files (**sa5cli** on Windows). Once this is done, simply adding the install directory to your **PATH** should allow you to successfully run **sa5** regardless of your current working directory. If the appropriate environment variable is not set, **sa5** will fail to run if its support files are not in the current working directory.

For directory storage and retrieval operations, ISC bundles with **sa5** a command line LDAP browser, **sa5ldap** (named **ldapsearch.exe** on Windows). This utility comes with its own man page, conf files, and sample scripts. It may be used to search an LDAP repository and retrieve certificates and/or CRLs for use with **sa5**.

FILES

Configuration Files

Command line options and/or distinguished name fields and a validity period for certificates may be placed one-per-line in a separate text file and specified on the command line using the **-@** option. See the supplied example *tsa5.cfg*.

The following (modified) X.509 tags are only recognized in such a configuration file:

CN= *common_name* (defaults to `-g` argument)
 T= *title*
 O= *organization*
 OU= *organizational_unit*
 OU2= *organizational_unit_2*
 L= *locality*
 SP= *state_or_province*
 C= *country*
 EM= *email_address*

A certificate validity period may also be specified using lines of the following form:

NB= *notBefore*
 NA= *notAfter*

where *notBefore* and *notAfter* are strings of exactly 12 digits in the following format:
 yyyyymmddHHMM

WARNING: You must specify the date/time with precisely twelve digits: two each for the month, day, hour, and minute, and four for the year. (Seconds are set to 0 and cannot currently be changed.) No sanity check is currently performed on the input values.

If no validity period is specified, the following default values are used:

NB= *current system date/time*
 NA= *one year hence*

Any line in the configuration file starting with a pound sign '#' is ignored as a comment.

WARNING: If it exists in the current directory, the special configuration file *sa5.cfg* is read automatically upon start up by **sa5**. Do not specify this configuration file using a `-@` option since that will cause it to be processed twice.

Key Type Parameter Files

The following **.kyp* files are currently supplied with **sa5**. Any of these filenames (without its *.kyp* extension) may be used as an argument for the `-t` option during key generation (`-g`).

<i>dsa-768.kyp</i>	768-bit DSA (FIPS 186, ANSI X9.30(1); ISC parameters)
<i>dsa-1024.kyp</i>	1024-bit DSA (FIPS 186, ANSI X9.30(1); ISC parameters)
<i>dsa-2048.kyp</i>	2048-bit DSA (FIPS 186-1, ANSI X9.30(1); ISC parameters)
<i>dsa-4096.kyp</i>	4096-bit DSA (FIPS 186-1, ANSI X9.30(1); ISC parameters)
<i>rsa-768.kyp</i>	768-bit RSA (PKCS#1, ANSI X9.31)
<i>rsa-1024.kyp</i>	1024-bit RSA (PKCS#1, ANSI X9.31)
<i>rsa-2048.kyp</i>	2048-bit RSA (PKCS#1, ANSI X9.31)
<i>rsa-4096.kyp</i>	4096-bit RSA (PKCS#1, ANSI X9.31)
<i>rsa-8192.kyp</i>	8192-bit RSA (PKCS#1, ANSI X9.31)
<i>us-b-163.kyp</i>	NIST ECC parameters for an elliptic curve over GF(2 ¹⁶³)
<i>us-b-233.kyp</i>	NIST ECC parameters for a curve over GF(2 ²³³)
<i>us-b-283.kyp</i>	NIST ECC parameters for a curve over GF(2 ²⁸³)
<i>us-b-409.kyp</i>	NIST ECC parameters for a curve over GF(2 ⁴⁰⁹)
<i>us-b-571.kyp</i>	NIST ECC parameters for a curve over GF(2 ⁵⁷¹)
<i>us-p-192.kyp</i>	NIST ECC parameters for a curve over a 192-bit GF(p)
<i>us-p-224.kyp</i>	NIST ECC parameters for a curve over a 224-bit GF(p)
<i>us-p-256.kyp</i>	NIST ECC parameters for a curve over a 256-bit GF(p)
<i>us-p-384.kyp</i>	NIST ECC parameters for a curve over a 384-bit GF(p)
<i>us-p-521.kyp</i>	NIST ECC parameters for a curve over a 521-bit GF(p)

Self-Decrypting Stubs

The following **.exe* files are currently supplied with **sa5**. Any of these filenames (without its *.exe* extension) may be used as an argument for the **-X** option during encryption to create an RC4-encrypted, self-decrypting archive for the indicated target system.

stbaixppcrc4.exe	IBM AIX/PPC (RC4)
stbhxparrc4.exe	HP-UX/PA-RISC (RC4)
stbirxmiprc4.exe	SGI IRIX/MIPS (RC4)
stblnxx86rc4.exe	Linux/i86 (RC4)
stbosxppcrc4.exe	MAC OS X/PPC (RC4)
stbsolsparc4.exe	Solaris/SPARC (RC4)
stbsolx86rc4.exe	Solaris/x86 (RC4)
stbtrualprc4.exe	Compaq Tru64/Alpha (RC4)
stbwinx86rc4.exe	Windows/x86 (RC4)

Security Policy Files

Unlike the GUI-based versions of SecretAgent, the **sa5** command line cannot use the security policy files produced by the Windows-based PolicyAgent program. However, beginning with release 5.2.8, **sa5** does allow a system administrator to control its behavior (*e.g.*, by disabling various features and/or enforcing certain configuration settings) simply by placing a special "security policy file" named *sa5so.cfg* in the root directory of the UNIX file system. A sample *sa5so.cfg* file is provided with the **sa5** distribution and contains documentation on its configuration.

Using the *sa5so.cfg* security policy file, a Security Officer may specify:

- optional key recovery agents and/or groups
- a trusted directory containing CA certificates for path validation
- a trusted directory containing CRLs

He may also restrict, enforce or prohibit use of the following program features:

- key generation
- the generation of self-signed certificates
- the use of self-signed certificates
- the use of self-decrypting archives
- certificate path validation
- CRL usage
- permitted symmetric ciphers
- allowed compression schemes during encryption
- ciphertext archive output formats
- the creation of multiple file archives
- the creation of separate file archives
- prevent users from signing and encrypting files
(will still allowing encrypt-only or sign-only operations)
- secure erasure of sensitive files

To display the security policy restrictions currently in effect on your system, use the command:

```
sa5 -w2 -O
```

Environment and Special Files

Upon startup, the program looks for a configuration file named *sa5.cfg* in the current directory. If such a file exists, it is processed before all other command line arguments; no warning is issued if the file doesn't exist. Additional configuration files may be explicitly specified by using the **-@** option.

sa5/UNIX can use the environment variable **SA\$HOME** (**RANDDIR** on Windows) to determine where its **.kyp* files and self-decrypting stub files (**.exe*) are located. If this variable is not set, **sa5** assumes that all of these files reside in the current directory.

The environment variable **SA\$PWD** can be used to provide a password (in lieu of the **-p** option) when generating keys, decrypting, signing, or exporting a PKCS#12 file.

DATA RECOVERY

The simplest data recovery scheme just uses one or more designated key recovery agent (KRA) certificates as additional *virtual* recipients for all messages encrypted by a user. In this scenario, a message can be decrypted by an individual KRA just as any other authorized recipient would decrypt it. Any number of KRA keys can be used in such a scheme and, if desired, their use can be enforced by security policy settings.

A more complex approach supported by **sa5**, called a "secret sharing scheme," employs one or more virtual recipient keys whose corresponding private key are "split" between two or more KRAs with the same DSA/DH or ECC key types. (Secret sharing of RSA keys is not supported in the current release!)

To see how secret sharing works in the simplest case, assume that we have two KRA public keys (with common parameters) in *kra1.cer* and *kra2.cer*, and that a user has (either voluntarily or due to security policy settings) added `-e kra1+kra2` to his command line when encrypting a file. In an emergency, a security officer (SO: a trusted party possibly different from any of the KRAs), can request that the two KRAs jointly decrypt the file by acting according to the following protocol:

1. The SO obtains the ciphertext archive, say *archive.sa5*, and extracts its "header" using the command:

```
sa5 -o arch.sah archive.sa5
```

2. The SO transmits only the header file, *arch.sah*, to the individual KRAs with a request that they perform their role in the data recovery process. (Proof of the SO's authority to make such a request may be required as a matter of policy.)

3. Upon receipt of a *.sah* header file from an authorized SO, a KRA would "decrypt" it to produce a *.sar* "recovery" file. For this step, *kra1* runs the command:

```
sa5 -d kra1 -p kra1pwd arch.sah
```

and renames the *arch.sar* output file to *arch1.sar*. Similarly, *kra2* runs:

```
sa5 -d kra2 -p kra2pwd arch.sah
```

and renames his *arch.sar* output file to *arch2.sar*.

4. The individual KRAs *securely* transmit their *.sar* files back to the SO.
5. The SO uses a complete set of *.sar* files from a KRA group to decrypt the original ciphertext archive:

```
sa5 -a arch1.sar,arch2.sar archive.sa5
```

As with decryption by an ordinary recipient, all plaintext files are recovered from *archive.sa5* with their original filenames intact. Although we have illustrated the key recovery protocol using a *.sa5* archive, the same approach also works with auto-encrypted *.saa* and self-decrypting *.exe* files as long as they are created with key recovery in effect.

Actually, anyone in possession of the ciphertext file *archive.sa5* may perform the non-cryptographic operation in step 1; a password is not required. Since the *.sah* file produced in this step contains only the public key wrapped random session keys extracted from *archive.sa5*, *archive.sah* is no more sensitive than the ciphertext file itself. For this reason, *.sah* files may, if necessary, be transmitted in the clear over the same insecure communications channels as the original ciphertext archive.

WARNING: While individual *.sar* files (which contain the *.sa5* archive's session key *partially* unwrapped by a *single* KRA) cannot be used to decrypt the archive, when combined, the *.sar* files corresponding to a complete set of split KRA keys *can* be used to decrypt the archive. Consequently, *.sar* files should *not* be transmitted in the clear over an insecure channel. If an insecure channel must be used, we recommend that each KRA encrypt their *.sar* file for the SO prior to transmission in step 4. (Of course, this means the SO must perform some preliminary decrypt operations in step 5 to recover the plaintext *.sar* files before actually decrypting the *.sa5* file of interest.)

For auditing purposes, the KRAs might also be required to sign, as well as encrypt, their *.sar* files when transmitting them back to the SO. If he stores the signatures in a secure database, the SO can later

provide proof that the agents did indeed participate in the emergency data recovery operation.

NOTES: Any number of (non-RSA) keys of the same size and type can be combined into a single virtual recipient (by +'ing their respective *.cer* files together) to create a secret sharing group for key recovery purposes. As many independent key recovery "groups" as desired can be employed for redundancy purposes.

To enforce key recovery, place a `-e` option with one or more (split or individual) key recovery agent keys in a configuration file and insist that your users include that *.cfg* file on every `sa5` command line along with their own second `-e` option specifying ordinary recipients. (The special configuration file *sa5.cfg* is ideally suited for this purpose.)

What we're offering here should really be referred to as "message recovery," and not "key recovery." The actions of the KRAs only allow the SO to recover the random session key used to encrypt a *particular* archive. This is *not* a private key escrow scheme and at no time does our approach require the "recovery" of a user's private key. Each message must be recovered independently and the recovery of one message does not compromise the security of any other message encrypted for the same set of recipients and/or key recovery agents.

DIAGNOSTICS

In general, a return code of 0 indicates success; a non-zero return code indicates that some processing error has occurred. The most common error codes are tabulated below.

- 1 invalid command line
- 0 success; no errors or warnings detected
- 1 bad handle; internal error; possibly a memory allocation issue
- 2 file not found; a required file could not be found; common reasons: missing KYP file, missing self-decrypting stub, SA5HOME not properly set, specified input file does not exist
- 3 invalid signature; the digital signature cannot be validated, possibly because the data file has been corrupted or modified
- 4 invalid key; cannot decrypt session key or key is improperly formatted; may also be reported if private key is missing or invalid during a signature operation; prior to the 5.8.0 release, this error was also returned when attempting to decrypt with a non-recipient's private key
- 5 invalid key file; when attempting to change the password on a private key file, this error code indicates that the input private key file cannot be opened or the old password is invalid
- 9 invalid password; the supplied password is invalid and cannot be used to unwrap the specified private key or self-decrypting archive
- 10 invalid private key; the private key cannot be unwrapped or is invalid and cannot be used for the current operation
- 11 invalid key type, or KYP file not found
- 12 cannot write file; an output file could not be created or written, possibly because it already exists and `-q` has not been specified, or `-i` was specified and the user decided not to overwrite the file; the file may be read only or the user may not have write permission in the necessary directory
- 13 invalid file format; a corrupt file, or file with an unknown format, compression, or encoding specification has been encountered
- 16 invalid data format; a certificate, certificate request, CRL, or other PDU (protocol data unit) could not be successfully parsed
- 21 no random seed; the pseudorandom number generator could not find the appropriate registry entry or random seed file (releases 5.6 and higher do not output this error, but running `sa5 -R` with an earlier release will reinitialize the necessary random seed object)
- 23 no recipient; no recipient certificates were specified for the encryption operation

- 24 unknown; an unknown error occurred or an unexpected input was encountered
- 25 invalid certificate; one or more of the certificates specified for the current operation was invalid (*i.e.*, it has expired, lacks the proper keyUsage attribute, is of an unknown key or algorithm type, or is otherwise invalid)
- 26 unsupported algorithm; the requested symmetric cipher or public key algorithm is not supported
- 28 required data not found; additional information is needed (for example, verifying a detached CMS signature requires that the original data file be provided as an additional input)
- 29 certificate cannot be used for encryption; certificate's keyUsage attribute or local security policy prohibits its use for this operation
- 30 certificate cannot be used to sign; certificate's keyUsage attribute or local security policy prohibits its use for this operation
- 31 cancelled; the user or system aborted the pending operation, most commonly due to a decision not to overwrite an existing file ('-q' always overwrites, '-i' prompts the user)
- 32 invalid PKCS#12 file; the specified PKCS#12 file could not be processed or the supplied password is incorrect
- 34 expired certificate; one or more of the certificates specified for the current operation has expired
- 36 invalid or incorrect issuer; an attempt was made to validate a certificate against the wrong issuer certificate
- 37 certificate revoked; an implicitly or explicitly referenced certificate has been revoked and cannot be used for the current operation
- 38 CRL out of date; an implicitly or explicitly referenced CRL is out of data and should be updated before retrying the current operation
- 39 invalid encoding; an input file's encoding type is not supported or the file is corrupt
- 40 unsupported compression mode; an input file's compression scheme is not supported or the file is corrupt
- 41 you are not a recipient; the supplied decryption key (or keys) does not match any recipient of the specified archive
- 43 cannot parse CRL; an implicitly or explicitly referenced CRL cannot be successfully processed
- 44 invalid serial number
- 45 unsupported hash type; the user has specified an unsupported algorithm id or an invalid algorithm id has been encountered in an input file
- 46 unknown cipher; the user has specified an unsupported algorithm id or an invalid algorithm id has been encountered in an input file
- 47 invalid message digest; the user has specified an unsupported algorithm id or an invalid algorithm id has been encountered in an input file
- 48 can't find self-decrypting stub; the specified self-decrypting stub could not be found, possibly due to a failure to correctly set the **SASHOME** environment variable

SEE ALSO

From time to time software patches and updated documentation are posted to the SecretAgent support pages on the ISC website:

<http://www.infosecorp.com/support/>

AUTHORS

SecretAgent is a direct descendent of Crypt Master, a hybrid file encryption utility designed and implemented by Mike Markowitz and Roger Schlafly in 1983. SecretAgent 1.0 was released by ISC in 1990; release 5.0 first shipped in 1997.

The current SecretAgent project team consists of the following engineers (who handle all on-going design, development, documentation, testing and maintenance tasks):

Randy Fulara	Roger S. Schlafly
Michael Gaspar	Jonathan C. Schulze-Hewett
Michael J. Markowitz	Pauline W. Tang

ACKNOWLEDGEMENTS

We are very grateful for the significant contributions to the SecretAgent project in the 1990s made by the following software engineers:

Navin Setia	David Sowinski
Mark Smith	Darian Woodford

We also wish to thank the following individuals for their input into the design, development, documentation, or testing of SecretAgent, or for their support of programming team members. (Our apologies for any inadvertent omissions.)

Ruban Ambrose	Susan Markowitz
Mark Audino	Andy McDermott
James Bendall	Scott Mitchell
Bill Bialick	Erv Morton
Gena Brown	Rich Nicholas
Glenn Brown	Art Nichols
John Burgess	Wes Nicolls
Mary Casey	Tom Patterson
John Centafont	Bob Perrey
David Dalkowski	Ben Phillips
Dave DeVita	Monette Respress
Rob Dobry	Joe Roth
Jim Dolphin	Bevan Rowley
Mark Etzel	Joe Schatz
Marnie Euteneuer	Julie Schlafly
Eva Jun	Mike Schimpf
Bob Forman	Bruce Schneier
Bill Franklin	Charlotte Schulze-Hewett
Elise Fulara	Stacy Seidl
Bill Gibson	Dave Sinclair
Erik Graham	Miles Smid
Mike Green	Thomas Venn
Jim Hunt	Mark Wallace
Jeff Kalibjian	Ori Wolman
Joe Kelly	Roger Woods
Steve Kowalski	Wei Wu
Sheshu Madabushi	

TECHNICAL SUPPORT

Software updates and technical release notes are available here:

<http://www.infosecorp.com/products/sacli/support.htm>

Technical support questions, bug reports, and comments may be submitted via e-mail to the following address:

tech@infosecorp.com

Information Security Corporation*Administrative & Marketing Office*

1141 Lake Cook Road, Suite D

Deerfield, IL 60015

Phone: 847-405-0500

Fax: 857-405-0506

Information Security Corporation*Research & Development Office*

1011 Lake Street, Suite 212

Oak Park, IL 60301

Phone: 708-445-1704

Fax: 708-445-9705

LEGAL INFORMATION

SecretAgent is a registered trademark of Information Security Corporation.

SA5, ALL KYP FILES, SELF32.EXE, AND THIS DOCUMENTATION ARE PROPRIETARY. THEY ARE FURNISHED UNDER A LICENSE OR NON-DISCLOSURE AGREEMENT. PLEASE TREAT THEM AS CONFIDENTIAL. NO FILES INCLUDED WITH THIS PACKAGE MAY BE REDISTRIBUTED BY YOU. IF YOU NEED TO REDISTRIBUTE THESE FILES WITH YOUR APPLICATION PLEASE CONTACT AN INFORMATION SECURITY CORPORATION ("ISC") SALES REPRESENTATIVE TO OBTAIN A REDISTRIBUTION LICENSE.

Information in this document is subject to change without notice and does not represent a commitment on the part of Information Security Corp. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the prior written permission of Information Security Corp.

SecretAgent is commercial computer software and, together with any related documentation, is subject to the restrictions on U.S. Government use as set forth below.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 52.227-7013. "Contractor/manufacturer" is Information Security Corporation, 1141 Lake Cook Road, Suite D, Deerfield, IL 60015, U.S.A.

LIMITATION OF REMEDIES: ISC's entire liability and your sole and exclusive remedy under this license and related to your use of the Software shall be:

1. The replacement of any CD-ROM not meeting ISC's limited warranty and which is returned to ISC or an authorized ISC dealer with a copy of your receipt; or
2. If ISC or the authorized dealer is unable to deliver a replacement CD-ROM which is free from defects in material or workmanship for the stated purpose, you may terminate this license agreement by returning the Software within ninety (90) days, and your money will be refunded.

IN NO EVENT WILL ISC BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE SUCH SOFTWARE EVEN IF ISC OR AN AUTHORIZED ISC DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

Copyright(c) 1991-2005 Information Security Corp. All rights reserved.