NAME

rabbitsign − sign applications for TI graphing calculators

SYNOPSIS

rabbitsign [*options*] [**-o** *appfile*] [**-k** *keyfile*] *hexfile ...*

rabbitsign [*options*] [**-k** *keyfile*] **-c** *hexfile ...*

DESCRIPTION

rabbitsign is an implementation of Texas Instruments' Rabin and RSA signing algorithms, as used on the TI-73, TI-83 Plus, TI-84 Plus, TI-89, and TI-92 Plus graphing calculators. These algorithms are used to sign Flash applications and operating systems so that the calculator can recognize them as valid.

rabbitsign, like Texas Instruments' official signing programs, needs a private key (a pair of large prime numbers) to sign apps. In order for the app to be accepted, the corresponding public key (their product) must be present on the calculator. As of this writing, the ''shareware'' private key number 0104, used for signing applications for the TI-83 Plus and TI-84 Plus, is available through TI's SDK. Unfortunately, the OS signing keys, as well as the app signing keys for the TI-73, TI-89, and TI-92 Plus, have not been released, which means that only TI can sign apps and OSes for those calculators.

OPTIONS

-a Attempt to match the output of Peter-Martijn Kuipers' **appsign** program, for testing purposes. The resulting output file will have Unix-style line termination, and hence will not be compatible with all programs. This option is not recommended for ordinary use.

-b Assume input files are raw binary files. If this option is not given, the file type is detected automatically.

-c Do not sign apps; instead, check that the signatures of the specified apps are valid. Exit status is 0 if all apps are valid, 1 if one or more apps fail, or 2 if there is a non-mathematical error.

-f Ignore non-fatal errors, and force the application to be signed if possible. (All of these messages are there for a reason, though, and chances are that if your app generates any of them, it will also either fail validation or crash the calculator. You have been warned.)

-g Write the output file in GraphLink ''TIFL'' format. (By default and for historical reasons, apps and OSes for the TI-73 and TI-83 Plus are written in plain TI Hex format instead; you can use **packxxk**(1) to convert these files into TIFL format.) Apps and OSes for the TI-89 and TI-92 Plus are always written in TIFL format. See **APPLICATION FILE FORMATS** below for more information.

-k *keyfile*
Read signing and/or validation keys from the given file. This file must be in one of the formats used by TI's SDK tools. (See **KEY FILE FORMATS** below.) By default, **rabbitsign** searches for the key named in the app header (for example, 0104.key for ''shareware'' TI-83 Plus apps.)

-K *id* Search for the key with the given *id* (a small hexadecimal number) rather than the ID specified in the app header.

-n Attempt to sign the program as-is, without modifying the header. (This option may not produce a file that the calculator will actually accept; it is intended for testing and special-purpose signatures, not for ordinary app signing.)

-o *outfile*
Specify the output file. By default, output files are named by taking the name of the input file, removing any suffix, and adding a '.app' or '.8xk' suffix depending on whether **-g** is specified. (If the input file already has a '.app' or '.8xk' suffix, '-signed' is inserted, so 'myapp.8xk' becomes 'myapp-signed.8xk'.)

If '-' is specified as an input file, that indicates the standard input, and the signed result is written by default to the standard output.

-p Fix the app pages header. This is not done by default because an incorrect pages header is generally a sign of a much more serious problem.

-P If the application ends close to a page boundary, add an additional page to hold the signature. (Application signatures on the TI-73 and TI-83 Plus are not allowed to span a page boundary.) Keep in mind that this is extremely wasteful, as it consumes 16384 bytes of Flash to hold approximately 69 bytes of data; if your application is in this situation, you should consider trying to reduce its size slightly, or alternatively, adding more data to take advantage of the extra Flash page.

-q Do not print non-fatal warning messages.

-r Re-sign a previously signed app (i.e., discard the previous signature before signing.)

-R *n* For signing TI-73 and TI-83 Plus applications, use root number n ($0 \le n \le 3$) rather than the default, root number 0. All four roots are valid, though distinct, signatures, so this option is mainly for debugging.

Root 0 is computed so as to be congruent to $m^{\left(\frac{p+1}{4}\right)}$ modulo p and $m^{\left(\frac{q+1}{4}\right)}$ modulo q. Root 1 is the negation of root 0 modulo p, root 2 the negation modulo q, and root 3 the negation both modulo p and modulo q.

This option has no effect when signing OSes or TI-89/92 Plus applications, which use the RSA algorithm rather than Rabin.

-t *type* Explicitly specify the type of program (e.g., '8xk' for a TI-83 Plus application, or '73u' for a TI-73 operating system.) The default behavior is to infer the program type from the name of the input file. (If the input file does not have a recognized suffix, the type is guessed based on the contents of the program header.)

-u Disable automatic page detection, and assume input files are unsorted. This means that page boundaries must be defined explicitly. (See **APPLICATION FILE FORMATS** below.) This option has no effect in binary (**-b**) mode.

-v Be verbose; print out the names of apps and their signatures as they are signed. Use **-vv** for more detailed information about the computation.

--help Print out a summary of options.

--version
Print out version information.

APPLICATION FILE FORMATS

Intel Hex

Intel hex is a standard ASCII file format used by many PROM programmers, and a common assembler output format. Each line (or ''record'') consists of the following:

:*NNAAAATTDDDDDD...CC*

: The first character of the line is a colon; this is just used to identify the format.

NN The next two characters are the number of data bytes on this line, written in uppercase ASCII hexadecimal.

AAAA The next four digits are the address of the data on this line. **rabbitsign** only looks at the low 14 bits of this value.

TT These two digits identify the ''type'' of the record. Intel hex defines several record types for various addressing models; the only types which are meaningful for TI calculators are types 00 (ordinary data) and 01 (end of file.)

DD... 2**NN* hex digits follow, the actual data.

CC Finally, the inverted checksum is added, so that adding up all the bytes on the line gives a total of zero modulo 256. (As an extension, **rabbitsign** permits you to use two uppercase 'X's in place of a checksum.)

Since the address is only 16 bits, this format cannot unambiguously represent applications larger than 64 kilobytes. To enable multi-page applications to be created, **rabbitsign** attempts to detect page boundaries automatically, starting a new page for each field with a zero address. Thus to create a multi-page app, you can simply concatenate several Intel Hex files, e.g.

```
cat page0.hex page1.hex | rabbitsign - -o complete.app
```

This will only work if the records are correctly sorted within each page. (If the assembler does not generate records in order, you can sort the fields yourself using a command such as 'sort -k1.8,1.9 -k1.4,1.7'.)

To turn off this automatic page detection, use the **-u** option. In this case, you must define page boundaries explicitly using the TI Hex format.

TI Hex

''TI'' hex is an extension to Intel hex which provides unambiguous representation of multi-page apps. It adds records of the form

:0200000200*NNCC*

which indicate that subsequent data is placed on relative page number *NN*. (Keep in mind that the TI-83 and 84 Plus install applications ''backwards,'' so if relative page 0 is stored on absolute page 69, relative page 1 will be stored on absolute page 68, and so forth.)

Some assemblers can generate multi-page data using type 4 records instead of type 2; **rabbitsign** treats these as equivalent to type 2.

For compatibility with other software which makes certain assumptions about the format, **rabbitsign** will write 32 bytes per line, and will end lines with a DOS-style carriage return and line feed.

GraphLink TIFL

The standard (newer) GraphLink format consists of a 78-byte binary header which is added to the start of a hex or binary file. The contents of this header are as follows:

0x00-07
The string '**TIFL**'.

0x08 The major version number (''App ID'') of the application.

0x09 The minor version number (''App Build'') of the application.

0x0A Flags; apparently intended to indicate whether the contents of the file are binary (0) or TI Hex (1). This value is not, however, set consistently by other software.

0x0B ''Object Type'' field; apparently indicates something about the type of data. Set to 0x88 in most TI-73 and TI-83 Plus files; set to 0 in TI-89 and TI-92 Plus files.

0x0C-0F
Binary-coded decimal month (one byte), day (one byte), and year (two bytes, big endian) when the app was signed.

0x10 Length of the app's name.

0x11-18
: Name of the app.

0x19-2F
: Reserved, always set to zero.

0x30 Calculator type (0x73 = TI-83 Plus, 0x74 = TI-73, 0x88 = TI-92 Plus, 0x98 = TI-89.)

0x31 Type of data (0x23 = OS upgrade, 0x24 = application, 0x25 = certificate.)

0x32-49
: Reserved, always set to zero.

0x4A-4D
: Little endian length of the following data (the length of the hex file, not the on-calculator size of the application.)

There also exist multi-part TIFL files, which simply consist of two or more TIFL files concatentated together. (For instance, a software license agreement or a certificate file can be attached to an application.) **rabbitsign** handles these files in a limited way: it will read only the first section with a recognized data type, ignoring any other data in the file. **rabbitsign** cannot create multi-part TIFL files, but they can be created using **packxxk**(1) (or simply using **cat**(1).)

Binary

rabbitsign can also read binary app files; they are assumed to be contiguous, so when signing a multi-page app, each page except the last must be filled to a full 16k.

KEY FILE FORMATS

Key files contain the data needed for signing and validating applications. (The portion of the key file used for validating is known as the ''public'' key; the portion used for signing is the ''private'' key. Only the public key is stored on the calculator itself.) Official key files from TI come in two varieties, known as ''Rabin'' and ''RSA'' formats. Note that **rabbitsign** currently supports using Rabin-type key files to generate RSA signatures, but not the other way around.

Rabin key format

The Rabin key file format is typically used for TI-73 and TI-83 Plus application signing keys. It consists of three lines, each containing a big integer. The first line is the public key, *n*; the second and third are its two factors, *p* and *q*.

Each line begins with two hexadecimal digits, giving the length of the number in bytes, followed by the bytes themselves, written in hexadecimal in little-endian order.

RSA key format

The RSA key file format is typically used for TI-89 and TI-92 Plus application signing keys. It consists of three lines: the key ID, the public key *n*, and the signing exponent *d*. (*d* is the inverse of the validation exponent, 17, modulo $\phi(n)$, and thus calculating *d* is computationally equivalent to factoring *n*.)

The key ID is a short hexadecimal number, which should match the contents of the 811x header field. The numbers *n* and *d* are written as big integers, as in the Rabin key format.

FILES

/usr/local/share/rabbitsign/*.key
: Private key files which will be used if the requested key is not found in the current directory.

BUGS

Who needs them?

rabbitsign accepts keys, applications, and even file names which cause TI's programs to crash or generate invalid signatures.

Some apps which come very close to filling the last page may not be usable with TI-Connect. This is a bug in TI-Connect. Try TiLP.

rabbitsign does not always generate the same signature as do TI's programs. This is not a bug; it is simply due to the differences in implementation. There are in fact four valid signatures for every application hash; all four are accepted by the calculator.

If you encounter a valid app which **rabbitsign** is unable to sign, or worse, generates an invalid signature, this is a serious bug, and I would like to know about it.

SEE ALSO

packxxk(1), **rskeygen**(1)

AUTHOR

Benjamin Moody <floppusmaximus@users.sf.net>