and strong awareness of the issues. Again, if any such bugs are suspected in the future, enabling the additional warning flags and scrutinizing the warnings would be in order.

To be clear, I believe qpdf to be well-behaved with respect to sizes and offsets, and qpdf's test suite includes actual generation and full processing of files larger than 4 GB in size. The issues raised here are largely academic and should not in any way be interpreted to mean that qpdf has practical problems involving sloppiness with integer types. I also believe that appropriate measures have been taken in the code to avoid problems with signed vs. unsigned integers from resulting in memory overwrites or other issues with potential security implications, though there are never any absolute guarantees.

7.6. Encryption

Encryption is supported transparently by qpdf. When opening a PDF file, if an encryption dictionary exists, the ***QPDF*** object processes this dictionary using the password (if any) provided. The primary decryption key is computed and cached. No further access is made to the encryption dictionary after that time. When an object is read from a file, the object ID and generation of the object in which it is contained is always known. Using this information along with the stored encryption key, all stream and string objects are transparently decrypted. Raw encrypted objects are never stored in memory. This way, nothing in the library ever has to know or care whether it is reading an encrypted file.

An interface is also provided for writing encrypted streams and strings given an encryption key. This is used by ***QPDFWriter*** when it rewrites encrypted files.

When copying encrypted files, unless otherwise directed, qpdf will preserve any encryption in force in the original file. qpdf can do this with either the user or the owner password. There is no difference in capability based on which password is used. When 40 or 128 bit encryption keys are used, the user password can be recovered with the owner password. With 256 keys, the user and owner passwords are used independently to encrypt the actual encryption key, so while either can be used, the owner password can no longer be used to recover the user password.

Starting with version 4.0.0, qpdf can read files that are not encrypted but that contain encrypted attachments, but it cannot write such files. qpdf also requires the password to be specified in order to open the file, not just to extract attachments, since once the file is open, all decryption is handled transparently. When copying files like this while preserving encryption, qpdf will apply the file's encryption to everything in the file, not just to the attachments. When decrypting the file, qpdf will decrypt the attachments. In general, when copying PDF files with multiple encryption formats, qpdf will choose the newest format. The only exception to this is that clear-text metadata will be preserved as clear-text if it is that way in the original file.

7.7. Random Number Generation

QPDF generates random numbers to support generation of encrypted data. Versions prior to 5.0.1 used *random* or *rand* from *stdlib* to generate random numbers. Version 5.0.1, if available, used operating system-provided secure random number generation instead, enabling use of *stdlib* random number generation only if enabled by a compile-time option. Starting in version 5.1.0, use of insecure random numbers was disabled unless enabled at compile time. Starting in version 5.1.0, it is also possible for you to disable use of OS-provided secure random numbers. This is especially useful on Windows if you want to avoid a dependency on Microsoft's cryptography API. In this case, you must provide your own random data provider. Regardless of how you compile qpdf, starting in version 5.1.0, it is possible for you to provide your own random data provider at runtime. This would enable you to use some software-based secure pseudorandom number generator and to avoid use of whatever the operating system provides. For details on how to do this, please refer to the top-level README.md file in the source distribution and to comments in *QUtil.hh*.

7.8. Adding and Removing Pages

While qpdf's API has supported adding and modifying objects for some time, version 3.0 introduces specific methods for adding and removing pages. These are largely convenience routines that handle two tricky issues: pushing inher-