passwords by doing the reverse operation: translating from correct in incorrect encoding of the password. This would enable qpdf to decrypt files using passwords that were improperly encoded by whatever software encrypted the files, including older versions of qpdf invoked without properly encoded passwords. The combination of these two recovery methods should make qpdf transparently open most encrypted files with the password supplied correctly but in the wrong coding system. There are no real downsides to this behavior, but if you don't want qpdf to do this, you can use the **--suppress-password-recovery** option. One reason to do that is to ensure that you know the exact password that was used to encrypt the file.

With these changes, qpdf now generates compliant passwords in most cases. There are still some exceptions. In particular, the PDF specification directs compliant writers to normalize Unicode passwords and to perform certain transformations on passwords with bidirectional text. Implementing this functionality requires using a real Unicode library like ICU. If a client application that uses qpdf wants to do this, the qpdf library will accept the resulting passwords, but qpdf will not perform these transformations itself. It is possible that this will be addressed in a future version of qpdf. The ***QPDFWriter*** methods that enable encryption on the output file accept passwords as strings of bytes.

Please note that the **--password-is-hex-key** option is unrelated to all this. This flag bypasses the normal process of going from password to encryption string entirely, allowing the raw encryption key to be specified directly. This is useful for forensic purposes or for brute-force recovery of files with unknown passwords.