PGP Software Developer’s Kit

Reference Guide

Version 1.7 Int.

PGP* Software Developer's Kit, Version 1.7.1 Int.

9-9-99. Printed in the EC.

Network Associates International BV.	+31(20)5866100
Gatwickstraat 25	+31(20)5866101 fax
1043 GL Amsterdam	http://www.nai.com
	info@nai.com

LIMITED WARRANTY

Limited Warranty. Network Associates Inc. warrants that the Software Product will perform substantially in accordance with the accompanying written materials for a period of sixty (60) days from the date of original purchase. To the extent allowed by applicable law, implied warranties on the Software Product, if any, are limited to such sixty (60) day period. Some jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

Customer Remedies. Network Associates Inc's and its suppliers' entire liability and your exclusive remedy shall be, at Network Associates Inc's option, either (a) return of the purchase price paid for the license, if any or (b) repair or replacement of the Software Product that does not meet Network Associates Inc's limited warranty and which is returned at your expense to Network Associates Inc. with a copy of your receipt. This limited warranty is void if failure of the Software Product has resulted from accident, abuse, or misapplication. Any repaired or replacement Software Product will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Outside the United States, neither these remedies nor any product support services offered by Network Associates Inc. are available without proof of purchase from an authorized international source and may not be available from Network Associates Inc. to the extent they subject to restrictions under U.S. export control laws and regulations.

NO OTHER WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, AND EXCEPT FOR THE LIMITED WARRANTIES SET FORTH HEREIN, THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" AND NETWORK ASSOCIATES, INC. AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, CONFORMANCE WITH DESCRIPTION, TITLE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM JURISDICTION TO JURISDICTION.

LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL NETWORK ASSOCIATES, INC. OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL OR EXEMPLARY DAMAGES OR LOST PROFITS WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF NETWORK ASSOCIATES, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, NETWORK ASSOCIATES, INC'S CUMULATIVE AND ENTIRE LIABILITY TO YOU OR ANY OTHER PARTY FOR ANY LOSS OR DAMAGES RESULTING FROM ANY CLAIMS, DEMANDS OR ACTIONS ARISING OUT OF OR RELATING TO THIS AGREEMENT SHALL NOT EXCEED THE PURCHASE PRICE PAID FOR THIS LICENSE. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

Table of Contents

Preface

The *PGP Software Developer's Kit Reference Guide, Version 1.7.1* is the primary reference source for using the PGP Software Developer's Kit ("PGPsdk"), which provides developers the functionality to readily add the PGP peer-reviewed cryptographic technology to their own applications. Because this is a reference manual, only a minimum of introductory or tutorial material is presented.

By using the PGPsdk as a part of your development effort, you can:

- develop products that are as secure as *PGP for Desktop Security, Version 6.5.1* (and optionally interoperating with it, where appropriate)
- easily develop, maintain, and use PGP cryptographic components in your application
- provide yourself and your customers with the confidence that comes from using the PGP trusted and peer-reviewed technology in your security protocols

The engineers at Network Associates, Inc., have used the identical PGPsdk supplied to external developers to produce *PGP for Desktop Security, Version 6.5.1*. Numerous excerpts from a sample application representing a greatly simplified version of *PGP for Desktop Security, Version 6.5.1* are included in this manual. In keeping with the PGP corporate policy of complete and open publication of source code for peer review, the final *PGP for Desktop Security, Version 6.5.1 Source Code* books (when available) will serve as the essential and definitive reference for developers using the PGPsdk for their own application development.

Audience

This book is intended for experienced software engineers and application developers who need to incorporate the PGP cryptographic functionality in their application, or are developing a product that needs to communicate with other applications that create or understand PGP-encrypted or cryptographically signed data. Since the initial release of the PGPsdk supports a *C* language Application Programming Interface (API), you should have *C* language experience to use this product.

If you are not familiar with basic cryptographic concepts, PGP recommends that you read *Applied Cryptography*, *Second Edition*, by Bruce Schneier (John Wiley & Sons, Inc., 1996). This volume is arguably the best introduction and general reference to cryptography currently available to the public. For additional readings on **cryptography** and cryptographic theory, see the short list of recommended readings at the end of this chapter, or the more extensive list in Appendix C, “References and Recommended Reading.”

How to use this guide

The *PGP Software Developer’s Kit Reference Guide* presents the PGP cryptographic functionality in a manner that corresponds to the organization of the PGPsdk Software Library. Several overview chapters appear first, and detail the basic concepts, organization, and functional divisions of the PGPsdk.

Following the overview chapters are detailed reference chapters for each functional division of the PGPsdk, which contain detailed descriptions of the functions in each functional division. The reference chapters include:

- an introductory overview of the functional division
- a list of the names of the associated *C* language header files
- tables containing `#define` and enumerated type constants and their descriptions
- *C* language code fragments for any associated datatypes and structures
- a logical ordering of the events and / or functions within the functional division

Each event description includes:

- an explanation of the event
- the data type and structures passed to / from the event
- the allowed `PGPO[ption]` values (if any)

Each function description includes:

- the function’s *C* language prototype
- argument descriptions
- an explanation of the function
- optional notable error codes
- optional notes, warnings, and tips on using the function

- optional sample code

The manual contains appendixes detailing:

- error codes
- recommended readings in cryptography

The manual concludes with:

- a glossary of cryptographic terms
- an index

Conventions used in this guide

Typographic conventions

C language code listings, reserved words, and names of data structures, fields, constants, arguments, and functions are shown in `Courier Font`.

Key terms or concepts appear in **boldface**, and are defined in the Glossary.

Notes, warnings, and tips conventions

Notes may contain:

- non-essential but useful and / or interesting information
- information that is essential for understanding the material presented

Warnings contain information that is essential to understand. Failure to do so could result in crashes and / or loss of data.

Tips contain information specifically intended to aid the PGPsdk developer in using the function to the best advantage.

How to contact Network Associates

Customer service

To order products or obtain product information, contact the Network Associates Customer Care department at (408) 988-3832 or write to the following address:

Network Associates International BV.
Gatwickstraat 25
1043 GL Amsterdam
Netherland

Technical support

Network Associates is famous for its dedication to customer satisfaction. We have continued this tradition by making our site on the World Wide Web a valuable resource for answers to technical support issues. We encourage you to make this your first stop for answers to frequently asked questions, for updates to Network Associates software, and for access to Network Associates news and encryption information.

World Wide Web	http://www.nai.com

Technical Support for your PGP product is also available through these channels:

Phone	+31(20)5866100
Email	tech-support-europe@nai.com

To provide the answers you need quickly and efficiently, the Network Associates technical support staff needs some information about your computer and your software. Please have this information ready before you call:

If the automated services do not have the answers you need, contact Network Associates at one of the following numbers Monday through Friday between 6:00 A.M. and 6:00 P.M.

Phone	+31(20)5866100

To provide the answers you need quickly and efficiently, the Network Associates technical support staff needs some information about your computer and your software. Please have this information ready before you call:

- Product name and version number
- Computer brand and model
- Any additional hardware or peripherals connected to your computer
- Operating system type and version numbers

- Network type and version, if applicable
- Content of any status or error message displayed on screen, or appearing in a log file (not all products produce log files)
- Email application and version (if the problem involves using PGP with an email product, for example, the Eudora plug-in)
- Specific steps to reproduce the problem

Network Associates training

For information about scheduling on-site training for any Network Associates product, call +31(20)5866100.

Comments and feedback

Network Associates appreciates your comments and feedback, but incurs no obligation to you for information you submit. Please address your comments about PGP product documentation to: Network Associates International BV., Gatwickstraat 25, 1043 GL Amsterdam, Netherland. You can also e-mail comments to tns_documentation@nai.com.

Year 2000 Compliance

Information regarding NAI products that are Year 2000 compliant and its Year 2000 standards and testing models may be obtained from NAI's website at http://www.nai.com/y2k.

For further information, email y2k@nai.com.

Development environment and API platform support

The PGPsdk, Version 1.7.1 binaries and public header files are supported on three major platforms: UNIX, 32-bit Windows, and Macintosh. While platforms and compilers other than those listed below may work with the PGPsdk (and some will be supported in future releases), the Version 1.7.1 release has only been verified as working with the following:

- UNIX platform and compiler support includes Solaris for Sparc, Linux x86, OpenBSD x86, and NetBSD x86 environments, each using the GNU *C* compiler.
- 32-bit Windows platform and compiler support includes those 32-bit environments using the Microsoft Visual C++ 5.0 compiler
- MacOS platform and compiler support includes MacOS Version 7.6 environments using the MetroWerks CodeWarrior Version 12.

Related documentation

The following documentation is available to help you install, configure, and get up to speed on the entire PGP product line.

- **An Introduction to Cryptography.** This guide is for anyone new to the science of cryptography. It is a high-level overview of the terminology, concepts, and processes used by PGP. It includes a section on security by PGP's creator, Phil Zimmermann.
- **PGP Installation Guide.** The Installation Guide describes how to install the following products:
  - **PGP Desktop Security.** Configuration techniques for PGP Desktop Security, including instructions on how to create a PGP Client installer with pre-configured settings, are included in the PGP Administrator's Guide.
  - **PGP Certificate Server.** Configuration techniques for the Certificate Server are included in the PGP Certificate Server Administrator's Guide.
  - **PGP Replication Engine.** Configuration techniques for the Replication Engine are included in the PGP Certificate Server Administrator's Guide.
  - **Policy Management Agent for SMTP.** Configuration techniques for the Policy Management Agent are included in the Policy Management Agent Administrator's Guide.
- **PGP Certificate Server Administrator's Guide.** The Administrator's Guide describes how to configure and administrate the PGP Certificate Server and PGP Replication Engine.
- **Policy Management Agent for SMTP Administrator's Guide.** The Administrator's Guide describes how to configure and administrate the Policy Management Agent.
- **PGP Desktop Security User's Guide.** The User's Guide describes how to use the email, file, and disk encryption utilities of PGP and PGPdisk.
- **PGPsdk User's Guide.** The SDK User's Guide describes how to use the PGP Software Developer's Kit.
- **PGP Product Source Code Books.** Philip Zimmermann, editor, Warthman Associates.

Recommended readings

Non-technical and beginning technical books

- Whitfield Diffie and Susan Eva Landau, “Privacy on the Line,” *MIT Press*; ISBN: 0262041677
  This book is a discussion of the history and policy surrounding cryptography and communications security. It is an excellent read, even for beginners and non-technical people, but with information that even a lot of experts don’t know.

- David Kahn, “The Codebreakers” *Scribner*; ISBN: 0684831309
  This book is a history of codes and code breakers from the time of the Egyptians to the end of WWII. Kahn first wrote it in the sixties, and there is a revised edition published in 1996. This book won't teach you anything about how cryptography is done, but it has been the inspiration of the whole modern generation of cryptographers.

- Charlie Kaufman, Radia Perlman, and Mike Spencer, “Network Security: Private Communication in a Public World,” *Prentice Hall;* ISBN: 0-13-061466-1
  This is a good description of network security systems and protocols, including descriptions of what works, what doesn't work, and why. Published in 1995, so it doesn't have many of the latest advances, but is still a good book. It also contains one of the most clear descriptions of how DES works of any book written.

Intermediate books

- Bruce Schneier, “Applied Cryptography: Protocols, Algorithms, and Source Code in C,” *John Wiley & Sons*; ISBN: 0-471-12845-7
  This is a good beginning technical book on how a lot of cryptography works. If you want to become an expert, this is the place to start.

- Alfred J. Menezes, Paul C. van Oorschot, and Scott Vanstone, “Handbook of Applied Cryptography,” *CRC Press;* ISBN: 0-8493-8523-7
  This is the technical book you should get after Schneier. There is a lot of heavy-duty math in this book, but it is nonetheless usable for those who do not understand the math.

- Richard E. Smith, “Internet Cryptography,” *Addison-Wesley Pub Co*; ISBN: 020192480
  This book describes how many Internet security protocols. Most importantly, it describes how systems that are designed well nonetheless end up with flaws through careless operation. This book is light on math, and heavy on practical information.

- William R. Cheswick and Steven M. Bellovin, “Firewalls and Internet Security: Repelling the Wily Hacker” *Addison-Wesley Pub Co*; ISBN: 0201633574
  This book is written by two senior researcher at AT&T Bell Labs, about their experiences maintaining and redesigning AT&T’s Internet connection. Very readable.

Advanced books

- Neal Koblitz, “A Course in Number Theory and Cryptography” *Springer-Verlag*; ISBN: 0-387-94293-9
  An excellent graduate-level mathematics textbook on number theory and cryptography.

- Eli Biham and Adi Shamir, “Differential Cryptanalysis of the Data Encryption Standard,” *Springer-Verlag*; ISBN: 0-387-97930-1
  This book describes the technique of differential cryptanalysis as applied to DES. It is an excellent book for learning about this technique.

PGPsdk Overview 1

Introduction

The PGPsdk consists of nine functional groups including, among others, **key management** functions, high- and low-level cryptographic functions, and pseudo-random number generation functions. Each group has a separately-compilable public header file that allows developers to include only the PGP cryptographic functionality that they want to impart to their applications. The more closely related header files are further grouped into twelve major functional areas. Each of these major functional areas is documented in its own chapter (Chapter 2 through Chapter 13).

Table 1-1. Public Header File Organization in This Document

Header File	Chapter
pgpOptionList.h	Chapter 2, "Key Management Functions"
pgpKeys.h	Chapter 3, "Option List Functions"
pgpGroups.h	Chapter 4, "Group Functions"
pgpCBC.h	Chapter 5, "Ciphering and Authentication Functions"
pgpCFB.h	
pgpEncode.h	
pgpHash.h	
pgpHMAC.h	
pgpPublicKey.h	
pgpSymmetricCipher.h	
pgpOptionList.h	
pgpFeatures.h	Chapter 6, "Feature (Capability) Query Functions"
pgpMemoryMgr.h	Chapter 7, "Utility Toolbox"
pgpPubTypes.h	
pgpSDKPrefs.h	
pgpUtilities.h	
pgpRandomPool	Chapter 8, "Global Random Number Pool Management Functions"
pgpUserInterface.h	Chapter 9, "User Interface Functions"
pgpKeyServer.h	Chapter 10, "Key Server Functions"

Header File	Chapter
pgpTLS.h	Chapter 11, “TLS (Transport Layer Security) Functions”
pgpSockets.h	Chapter 12, “Socket Functions”
pgpBigNum.h	Chapter 13, “BigNum Functions”
pgpErrors.h	Appendix A, “PGPsdk Error Summary.”
pgpPFLErrors.h	

Here are summaries of the chapters in the function reference sections of this book:

- Chapter 2, “Key Management Functions” Key management functions allow applications to create, sign, add, remove, search for, and check the validity of **keys** on disk-based or in-memory key rings. Also found here are functions to check and set property values for keys, according to the PGP **Web of Trust** model, as well as functions that import and export keys to files and buffers. The key management function prototypes are listed in the public header file `pgpKeys.h`.

- Chapter 3, “Option List Functions” Option list functions provide a flexible and extensible mechanism for presenting arbitrary option specifications and data to functions accepting this mechanism. Option lists may be persistent or local to the function accepting them, and so support modular establishment and combining of option groups. The option list function prototypes are listed in the public header file `pgpOptionList.h`.

- Chapter 4, “Group Functions” Group functions allow storing and manipulating persistent list of key IDs. The group management function prototypes are listed in the public header file `pgpGroups.h`.

- Chapter 5, “Ciphering and Authentication Functions” Algorithm-independent functions are provided for high-level cryptographic functions such as encrypting, decrypting, hashing, signing, and verifying messages. Not only are applications free of the details of the particular algorithms being used, but also new algorithms can be transparently incorporated as they become available. The high-level cryptographic function prototypes are listed in the public header file `pgpEncode.h`. The low-level cryptographic function prototypes are listed in the public header files `pgpCBC.h`, `pgpCFB.h`, `pgpHash.h`, and `pgpSymmetricCipher.h`, which appear as `#include` directives in `pgpEncode.h`.

- Chapter 6, "Feature (Capability) Query Functions" The present state of U.S. export law and the continuously evolving set of cryptographic standards, algorithms, and formats make the simultaneous existence of multiple versions of the PGPsdk a very real possibility, for example, a version intended for export may support signing but not encryption. The PGPsdk includes functions that return version numbers and the availability of specific features (capabilities), thus providing applications with a measure of version independence. The feature query function prototypes are listed in the public header file `pgpFeatures.h`.

- Chapter 7, "Utility Toolbox" The PGPsdk require miscellaneous utility functions such as memory management, context creation, file specification, preferences, and date / time functions. Additionally, this chapter documents a translation function that converts `PGPError` numeric codes to English language character strings. The utility function prototypes are listed in the PGPsdk public header files `pgpMemoryMgr.h`, `pgpPubTypes.h`, `PGPsdkPrefs.h`, and `pgpUtilities.h`.

- Chapter 8, "Global Random Number Pool Management Functions" Since the PGPsdk cryptographic functions require **random numbers** to operate correctly, the PGPsdk includes functions to manage a global pool of random numbers seeded from keystrokes and mouse movements. The **SHA-1 hash function** is used to distill entropy from incoming events and to spread it throughout the random pool. The random number generation function prototypes are listed in the public header file `pgpRandomPool.h`.

- Chapter 9, "User Interface Functions" The PGPsdk includes User interface elements such as passphrase and key selection dialogs which allow developers to present an interface consistent with the PGP product, if desired. These functions are available on the Windows and MacOS platforms only. The user interface function prototypes are listed in the public header file `pgpUserInterface.h`.

- Chapter 10, "Key Server Functions" The PGPsdk includes functions to facilitate communicating with both **HTTP** and **LDAP** key servers. The key server function prototypes are listed in the public header file `pgpKeyServer.h`.

- Chapter 11, "TLS (Transport Layer Security) Functions" The TLS functions provide a transport-layer independent means of encrypting and authenticating network communication. The TLS function prototypes are listed in the public header file `pgpTLS.h`.

- Chapter 12, "Socket Functions" The PGPsdk socket functions allow sophisticated PGPsdk developers further access to the functions that form the basis for secure communication between PGP client and server applications. The socket function prototypes are listed in the public header file `pgpSockets.h`.

- Chapter 13, “BigNum Functions” The PGPsdk includes a set of utilities for manipulating large, multiple precision integers (BigNums). The BigNum function prototypes are listed in the public header file `pgpBigNum.h`.

PGPsdk functionality

The PGP Software Development Kit (PGPsdk) allows software engineers and application developers to seamlessly incorporate the PGP cryptographic technology into such applications as e-mail package plug-ins, secure electronic interchange packages, and secure financial transaction packages. The PGP cryptographic technology consists of the following three basic cryptographic elements:

- key management
- ciphering (**encryption** / **decryption**)
- **authentication** (signing and verifying)

Key management functions:

- create and / or add keys
- remove keys
- search for keys meeting certain ownership and / or property criteria
- check the validity of disk-based or in-memory key rings
- check and / or set key property values
- create, delete, and modify logical groups of keys

Ciphering (encrypting / decrypting) functions:

- encrypt data or files
- decrypt data or files

Authentication (signing and verifying) functions:

- sign messages or data files
- verify the authenticity of messages or data files

Other functional areas include **pseudo-random number** generation, BigNum manipulation, utility, feature availability query, and key server access functions that:

- manage pseudo-random numbers seeded from mouse movements, keystrokes, and other events

- manipulate large integers, such as the large primes that form the basis of modern cryptographic keys
- manage memory
- specify files
- effect date / time conversion (platform dependent)
- indicate the availability of specific features within the PGPsdk
- convert error codes to readable strings
- communicate with and make requests of a remote key server and its associated key database(s).

The Application Programmer's Interface (API) to the PGPsdk consists of *C* language functions, and provides developers with a consistent interface and error handling protocols. These functions are organized into functional groups, and each group comprises a function reference chapter of this document (Chapter 2 through Chapter 12). Each of these chapters includes:

- an overview of the functional group
- a logical ordering of the functions within the group (as applicable)
- the function group's associated header file(s)
- a full description of each individual function

The full description of each function includes:

- a brief description of the function
- the function's *C* language prototype
- argument descriptions
- notes regarding use of the function
- sample code (as required)

To use the PGPsdk, simply incorporate calls to the PGPsdk functions into your *C* language application by using the function prototypes listed in the public header files supplied as part of the PGPsdk and including the necessary header files, and then linking with the supplied PGPsdk library binaries. PGP supplies two versions of the PGPsdk library binaries - a debug version and a non-debug version. Both versions perform essentially the same error checking, and report the same error return codes. The debug version additionally asserts itself on error conditions, and reports the errors to the default output destination (platform dependent).

Library binaries

The PGPsdk library binaries contain all of the functions described by the header file function prototypes, and link with your application. These libraries are distributed in both debug and non-debug versions, and have the following names on the following supported platforms:

MacOS

```
PGPsdkLib
PGPsdkNetworkLib
PGPsdkUILib
```

Win-32

```
PGP_SDK.dll
PGPsdKS.dll
PGPsdkNL.dll
PGPsdkUI.dll
```

Unix

```
libPGPsdk.a
libPGPsdkKeyServer.a
```

Note that the network library is required only for those applications that implement direct communication with a key server or implements transport layer security (see Chapter 10, “Key Server Functions”)

Thu user interface library is required only for those applications that implement PGP supplied user interface elements (see Chapter 9, “User Interface Functions”)

Data Type, constant, macro, and function name conventions

PGPsdk data types, macros, and functions have names beginning with `PGP`; PGPsdk constants have names beginning with `kPGP` (see “Summary of the PGPsdk Opaque Data Types”).

Most PGPsdk data types are opaque, that is, they are references to the actual data. These data types have names of the form:

```
PGPnameRef
```

where *name* describes the data type. Because these data types are opaque, a reference to one is not necessarily a pointer in the *C* language sense, and so they should never be de-referenced.

All of the PGPsdk opaque data types have special values to indicate that they are not referencing a valid instance. These values are useful for establishing an initial or default value, and have names of the form:

```
kInvalidPGPnameRef
```

The PGPsdk supports byte array data through use of the *C* language types `char[]` and `void[]`, as well as their associated pointer types `char*` and `void*`. While these basic types may or may not have implementational differences, they do have important PGPsdk-specific semantic differences:

- `char[]` and `char*` always denote `NULL` terminated byte arrays, that is, *C* language strings
- `void[]` and `void*` always denote arbitrary byte arrays that may coincidentally be `NULL` terminated.

PGPsdk constants have names of the form:

```
kPGPCategoryDescription
```

for example, `kPGPKeyPropCanSign`. `kPGP` is the constant data type prefix, `KeyProp` indicates that the constant belongs to the category that refers to key properties, and `CanSign` implies a boolean indicating whether or not the associated key is allowed to sign other keys.

PGPsdk macros and functions have names of the form:

```
PGPname
```

which is a very general format. However, there are several categories of functions that have noteworthy naming conventions and implied semantics:

Data Reference Macros

Macros having names of the form:

```
PGPnameRefIsValid
```

facilitate validation of opaque data types, and return a boolean value. Use of these macros is strongly encouraged, as they provide the PGPsdk developer with a guaranteed method for determining the validity of a data reference, while also maintaining its opacity.

PGPNew*Datatype* and PGPFree*Datatype*

`PGPNew`*`Datatype`* functions allocate a new, persistent instance of a PGPsdk opaque data types. The PGPsdk developer must eventually de-allocate the instance with the corresponding “free” function. For example, `PGPNewContext` allocates a new `PGPContextRef`, and `PGPFreeContext` de-allocates a `PGPContextRef`. Note that closely related PGPsdk opaque data types may share the same “free” function, for example, `PGPNewContextCustom` also uses `PGPFreeContext`.

PGPO*option*

`PGPO`*`option`* functions allocate `PGPOptionListRef` instances that are automatically de-allocated once they are used in an option list management function, for example, `PGPBuildOptionList`, or as a sub-option, for example:

```
PGPOSignWithKey( …, PGPOPassphrase( … ), … );
```

Other PGPsdk data types that have noteworthy implied semantics include:

PGPSize

`PGPSize` implies a length quantity, and further implies an in-memory context (similar to the *C* language pseudo-type `size_t`). Values associated with `PGPSize` items are in terms of the platform’s commonly used length quantity, which is almost always the 8-bit byte.

PGPFlags

`PGPFlags` items differs for other PGPsdk data types that assume enumerated values in that the associated values may be combined with boolean expressions to create masks, for example:

```
if  ( ( myFlags & ( kPGPKeyRingOpenFlags_Mutable |
              kPGPKeyRingOpenFlags_Create ) ) )
{
    /* features-are-available code */
}
```

PGPContext

The PGPsdk incorporates a global context / configuration mechanism for all PGPsdk functionality. The `PGPContext` data type replaces the many global variables used in previous PGP libraries, and thus provides a more robust and manageable application environment. Typically, an application will create a `PGPContext` at startup, use the context throughout its run, and finally free the context on exit.

The resultant `PGPContextRef` value is passed directly to most of the PGPsdk functions. However, some PGPsdk data types incorporate the `PGPContextRef` used to create them, and so the functions that accept these data types as arguments generally do not also require a `PGPContextRef` argument.

A `PGPContext` must *not* be freed until and unless all data items allocated with that context have already been freed. Failure to follow this protocol will not only result in memory leaks, but also precipitate application failures due to the associated context being invalid or incorrect.

IMPORTANT: The PGPsdk is thred-safe only through the use of different contexts in different threads. A single `PGPContextRef` cannot be safely used in multiple threads. It is the application developer's responsibility to enforce this semantic.

Most PGP opaque data types have an associated reference count of type …`RefCount`, which provides for simplified garbage collection. Upon creation of such a data type, its reference count is initialized to one. From that point, the PGPsdk automatically tracks the number of references to a particular resource, for example, a given key set may be referenced by any number of key lists and / or iterators. This not only results in a level of context independence, but also ensures that a resource's memory is released only when its last reference is deleted. The PGPsdk also provides functions to support manual adjustment of reference counts.

However, the automatic nature of the reference count management applies only to implied references. This means that the reference count of an underlying key set is automatically incremented whenever a key list is created from it, and is automatically decremented when that key list is freed. The PGPsdk developer is expected to adhere to the following basic rule:

> *All PGP opaque data types that are explicitly created (PGPNew… functions), copied (`PGPCopy`… functions), or have had their reference count manually incremented must be freed using the appropriate `PGPFree`… function.*

Memory management

Memory management within the PGPsdk is normally handled transparently by default functions analogous to the Standard *C* Library functions `malloc`, `dealloc`, and `realloc`. However, developers can override this behavior by specifying their own equivalent allocate, de-allocate, and reallocate functions (see the `PGPNewContextStruct` data type that is used by the `PGPNewContextCustom` function).

Generally speaking, any PGPsdk function having a name of the form:

`PGPNew…datatype…`

accepts a `PGPContext` reference, and allocates memory which the caller must explicitly de-allocate with the corresponding PGPsdk function having a name of the form:

`PGPFree…datatype…`

Some allocations within the PGPsdk do not have a working `PGPContextRef` from which to obtain a custom memory allocator (if any). If your application uses a custom memory allocator, be sure to set the default internal PGPsdk memory allocator with `PGPSetDefaultMemoryMgr()`.

Error codes

With several exceptions, PGPsdk functions return an error code (`kPGPError_…`) or `void`, and place any result values into output arguments. This convention allows for simple and consistent error checking. The PGPsdk provides the macros `IsPGPError` and `IsntPGPError` to test a function's return code. Essentially all PGPsdk functions that return an error code can return one or more of the following:

- `kPGPError_NoErr`
- `kPGPError_BadParams`
- `kPGPError_OutOfMemory`

These error codes are only listed for a function when their return has non-obvious or additional implications. Of course, a function that has no parameters cannot return `kPGPError_BadParams`, nor can a function that does not allocate memory return `kPGPError_OutOfMemory`.

PGPsdk API details and data structures -- Key management

Understanding how the PGPsdk key management functions perform their tasks requires understanding of several PGPsdk Version 1.5-specific concepts and data types. The following sections introduce the PGP key database, collections of keys from a key database, the construction of filters that in turn create collections of keys, ordered lists of keys from a collection of keys, and methods of iterating over an ordered list of keys.

A number of options is available for several of the key management functions, and each is defined as a function returning a `PGPOptionListRef` (see Chapter 5, "Option List Functions"). A special argument provided by the `PGPOLastOption` function *must* appear as the last argument to indicate the end of the list.

Key database

The PGP key database represents one or more key files, and can be thought of as a backing store for a key ring. It can be composed of any number of files on disk, or it can be entirely memory based. While the `PGPKeyDB` is a very important data type to understand, it is currently never exported, nor is there currently a user-visible reference type.

Every key in the system belongs to exactly one key database. Whenever a key is modified, its corresponding key database is also modified. While equivalent keys may exist simultaneously in several key databases, each instance is a distinct key from the point of view of the PGPsdk key management functions - each instance has a unique pointer, and so modifications to one will not affect any of the others.

Collections of keys in a key database

The `PGPKeySet` data type represents a subset (referred to as a *key set*) of exactly one key database, and may be thought of as a view onto that key database. The function `PGPOpenDefaultKeyrings` opens the caller's default key rings, which is conceptually a key database consisting of two files—the caller's **public key** and **private key** keyring files. The function then creates and returns a key set containing the full set of keys in that key database.

Any number of key sets may exist for a given key database (see the discussion on key filters in this chapter). For instance, one could create a key set that includes all keys, as well as a key set that includes only those keys signed by "Philip Zimmermann."

A key set is generally an "active" or a "live" view on a key database. To demonstrate what an active view is, consider a key set that is composed of all the keys that contain the name "Mark." Creating this key set with an active key filter and then adding a key containing name "Mark" to the associated key database results in that key being automatically and instantaneously added to the created key set, and vice versa.

Key filters

The PGPsdk allows the developer to construct very complicated key **filters** for operating on elements of the key database. These filters are built from **primitive key filters**, which in turn are created by the various `PGPNew…Filter` functions. These primitive key filters are generally of the form:

```
select all X that contain Y
```

A set of related functions allows negation, union, and intersection of primitive key filters, and so allows creation of key filters that implement arbitrary expressions such as

```
select all keys NOT containing “Phil” AND
    having keylengths longer than 1024 bits
```

Once the key filter is complete, the `PGPFilterKeySet` function applies the resultant key filter to a key set, yielding a new key set whose members satisfy the key filter criteria. Note that this resultant new key set may be empty.

Lists of keys in a key database

Key sets have no ordering – they are merely collections of keys. The `PGPKeyList` data type facilitates operations on key sets by imposing an ordering that may be based on any sortable data item or sub-structure within a key, for example, name or key ID. The function `PGPOrderKeySet` accepts a key set and a sort order specification, and yields a key list.

The `PGPKeyIter` data type implements iterating over a key list. Initially, it references the pseudo-element just before the first element in the key list, and then increments itself successively through each element of the key list. Most changes to a key list that occur while iterating are handled automatically. For example, inserting a new key causes the iteration to automatically “follow” the key it was working on. The `PGPKeyIter` data type also supports iteration over the sub-structures within the key, for example, iterating over the user ID structures of the key.

PGPsdk API details and data structures - Ciphering

Using the PGPsdk ciphering API

The PGPsdk Ciphering API has two high-level entry points - `PGPEncode` and `PGPDecode`. `PGPEncode` provides for all encrypting and signing functionality, while `PGPDecode` provides for all decrypting and signature verification functionality. Each function accepts a `PGPContextRef`, and a variable number of options that control the behavior of the function. The similarity of their prototypes is illustrated by the following examples:

```
PGPError PGPEncode( PGPContextRef pgpContext,
                    PGPOptionListRef firstOption,
                    ...,
                    PGPOLastOption( void ) );

PGPError PGPDecode( PGPContextRef pgpContext,
                    PGPOptionListRef firstOption,
                    ...,
                     PGPOLastOption( void ) );
```

A large number of options is available for both `PGPEncode` and `PGPDecode`, and each is defined as a function returning a `PGPOptionListRef`. Some options are suitable only for encoding operations, some options are suitable only for decoding operations, and some options are suitable for both operations (see Chapter 3, “Option List Functions”). A special argument provided by the `PGPOLastOption` function must appear as the last argument to indicate the end of the list.

Events and callbacks

The `PGPOEventHandler` option allows the calling application to request callbacks when various events occur, and to define a function (**event handler**) that is the target of the callback. While an event handler is usually not needed for encryption operations, it is often needed for decryption operations.

An event handler serves two purposes – it provides notification to the calling application that an event has occurred, and provides a mechanism for the calling application to affect processing (in a limited, pre-defined manner). Notification includes a `PGPEvent` reference which, depending on the type of event, provides detailed information about the cause of the event. The calling application can then respond appropriately, which may or may not affect the course of further processing. For certain events, the calling application can modify the processing context by invoking `PGPAddJobOptions`.

PGPsdk API details and data structures - Authentication

The PGPsdk performs authentication (signing and verification of messages) by using the supplied `PGPEncode` and `PGPDecode` functions. In the case of signing or verifying a message, the application invokes the appropriate `PGPO`... function(s), for example, `PGPOSignWithKey` and `PGPODetachedSig`, to perform the needed authentication function. In the case of authentication, the message is first passed through a hash function before being signed by the sender’s private key.

Hash Functions

The PGPsdk provides a number of hash functions (more commonly referred to as **hash algorithms**). Selection of a specific hash algorithm is sometimes implicit to the processing context; for example, **DSS** keys unequivocally use the SHA-1 hash algorithm. For other processing contexts, the `PGPOHashAlgorithm` function can be used to “manually” configure the context; for example, the function can force the use of the SHA-1 hash function in an **RSA** signature.

Key Management Functions 2

Introduction

The PGPsdk key management functions allow applications to create, sign, add, remove, search for, and check the validity of keys on disk-based or in-memory key rings. They also include functions that check and set property values for keys according to the PGP Web of Trust model, as well as functions that import and export keys to files and buffers.

A PGP key is always a signing key, and for certain algorithms is also an encryption key. If a sub-key is present, then it is always considered to be an encryption key. Some algorithms, for example the **Elgamal** variant of **Diffie-Hellman** require sub-keys since the base key is always considered to be sign-only. Other algorithms, for example RSA, do not support sub-keys, and for these the base key is used for both signing and encrypting.

Diffie–Hellman keys may have associated **additional recipient request keys**. When present, all messages encrypted to the base key should also be encrypted to each of the additional recipient request keys. The enforcement of this request is left to the application developer.

Diffie–Hellman keys may also have one or more associated **designated revocation keys**. A designated revocation key is empowered to revoke the subject key in the event the owner of the subject key is unable to revoke it—for example, if the private key has been lost of the passphrase forgotten.

A key may have any number of associated sub-keys, additional recipient request (ARR) keys, and user IDs. A user ID in turn may have any number of associated signatures.

Figure 2-1. Diffie-Hellman Key Structure

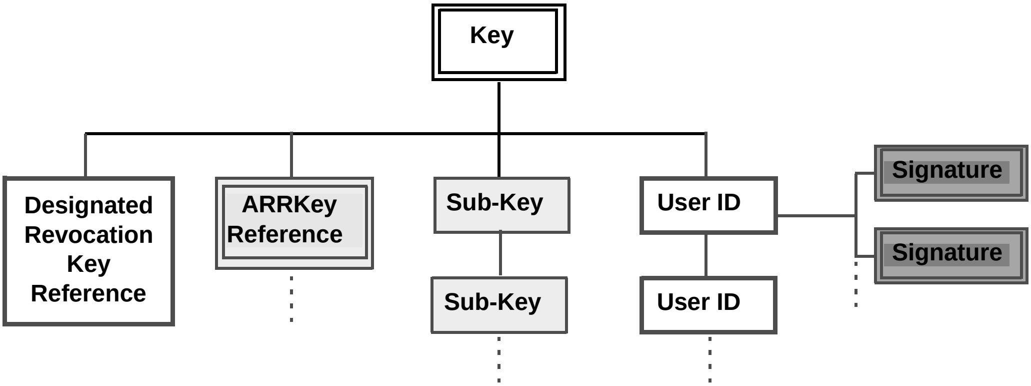
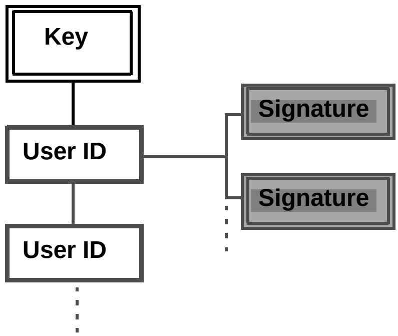


Figure 2-2. RSA Key Structure



Events and callbacks

A number of the key management functions allow the calling application to request callbacks to track the progress of the operation. Those functions that permit inclusion of a `PGPOEventHandler` option generally execute so quickly that an event handler is of limited benefit unless the key set involved is very large. Those functions that include an explicit event handler argument generally require a perceptible amount of execution time, regardless of the size of the key set.

An event handler serves two purposes – it provides notification to the calling application that an event has occurred, and provides a mechanism for the calling application to affect processing (in a pre-defined manner). Notification includes a pointer to a `PGPEvent` data type that, depending on the type of the event, provides detailed information about the cause of the event. The calling application can then respond appropriately, which may or may not intervene and affect the course of further processing. If the calling application wishes to intervene, then it can abort the job by returning an error code (a value other than `kPGPError_NoErr`). Additionally, depending on the type of event, it can modify the processing context by invoking `PGPAddJobOptions`.

All event handlers are declared as

```
PGPError myEvents( PGPContextRef pgpContext,
     PGPEvent *event,
     PGPUserValue userValue );
```

The `pgpContext` argument is the reference to the context of the function posting the event. The `event` argument references a `PGPEvent` data type as follows:

```
struct PGPEvent_
{
    PGPVersion      version;
    struct PGPEvent_*nextEvent;
    PGPJobRef       job;
    PGPEventType    type;
    PGPEventData    data;
};
typedef struct PGPEvent_ PGPEvent;
```

The `version` and `nextEvent` members are currently reserved for internal use. The `job` member is not applicable to key management functions. The `type` member identifies the event being posted, and recognizes `kPGPEvent_`... values. The `data` member is a `union` of the event-specific data structures, which are described with their corresponding event.

None of the key management functions currently support modification of the processing context by invoking `PGPAddJobOptions`.

Figure 2-3. (Sub-)Key Generation Event Sequence

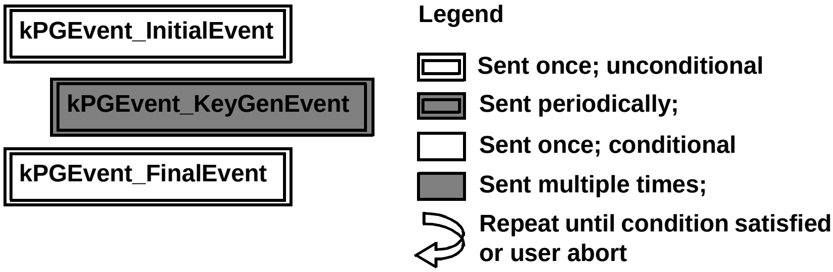
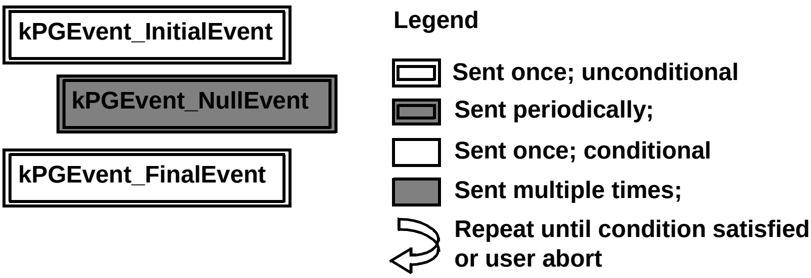


Figure 2-4. Key Set Operation Event Sequence



Key management events

kPGPEvent_InitialEvent

Sent before all other events. Implies entry to the function.

Data

`None`

Options

None

kPGPEvent_NullEvent

Sent during the course of key set import/export processing if explicitly requested with `PGPOSendNullEvents` (see `PGPExportKeySet` and `PGPImportKeySet`). Automatically sent during signature checking (see `PGPCheckKeyRingSigs`).

The event data allows the PGPsdk developer to determine the sending function's progress by way of its completion percentage. The event data members should be treated as relative, un-scaled quantities – they are not necessarily byte quantities or number-of-keys values. In all cases, the completion percentage is calculated as follows:

```
double completionPercent;

if  ( event->type = kPGPEvent_NullEvent )
{
    if  ( event->nullData.bytesTotal != 0 )
    {
        completionPercent = ( 100 *
            event->nullData.bytesWritten ) /
            event->nullData.bytesTotal;
    }
    else
    {
        completionPercent = 100;
    }
}
```

Data

```
typedef struct PGPEventNullData_
{
    PGPFileOffset bytesWritten;
    PGPFileOffset bytesTotal;
} PGPEventNullData;
```

kPGPEvent_KeyGenEvent

Automatically sent during the course of key and sub-key generation (see `PGPGenerateKey` and `PGPGenerateSubKey`).

The event data allows the PGPsdk developer to determine the progress of the key generation process. If the event handler returns an error, then the key generation process aborts.

The `state` value indicates the *approximate* state of the key generation process, and assumes the *character* values that were used by previous text-versions of PGP:

. selected value failed pseudo-primality test

/ all selected values failed pseudo-primality test; re-initializing the prime number generation environment

- selected value passed pseudo-primality test; further processing required

+ selected value passed pseudo-primality test; further processing required

***** selected value passed pseudo-primality test; processing for this phase is near completion.

space completion of this phase of (sub-)key generation. The actual number of phases varies from key to key, and has no fixed value or range

Data

```
typedef struct PGPEventKeyGenData_
{
    PGPUInt32state;
} PGPEventKeyGenData;
```

kPGPEvent_FinalEvent

Sent after all other events. Implies return from the function.

Data

None

Key ring management functions

PGPOpenDefaultKeyRings

Creates a key set that contains all of keys in the default public key and private key keyrings. Any **trust** information associated with the public key ring is included.

Syntax

```
PGPError PGPOpenDefaultKeyRings(
            pPGPContextRef pgpContext,
            PGPKeyRingOpenFlags openFlags,
            PGPKeySetRef *keySet );
```

Parameters

`pgpContext`	the target context
`openFlags`	the open options, which recognizes `kPGPKeyRingOpenFlags_`... values
`keySet`	the receiving field for the new key set

Flags

The open flags are interpreted as follows:

- `kPGPKeyRingOpenFlags_Mutable`
  `TRUE` if the resultant key set should be made modifiable; `FALSE` if the resultant key set should be made read-only.
- `kPGPKeyRingOpenFlags_Create`
  Set if the specified key ring file should be created if it doesn't already exist. Valid only if `kPGPkeyRingOpenFlags_Mutable` is also set.

Notes

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPOpenKeyRingPair

Creates a key set that contains all of the keys in the specified public and private key ring files. Any trust information associated with the public key ring is included.

See `PGPOpenDefaultKeyRings` for interpretation of the open flags.

Syntax

```
PGPError PGPOpenKeyRingPair (
             PGPContextRef pgpContext,
             PGPKeyRingOpenFlags openFlags,
             PGPFileSpecRef pubFileSpec,
             PGPFileSpecRef secFileSpec,
             PGPKeySetRef *keySet );
```

Parameters

`pgpContext`	the target context
`openFlags`	the open option flags value
`pubFileSpec`	the target public key ring file
`secFileSpec`	the target private key ring file
`keySet`	the receiving field for the new key set

Notes

For most applications, `PGPOpenDefaultKeyRings` provides all required functionality.

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPOpenKeyRing

Creates a key set that contains all of the keys in the specified key ring file.

See `PGPOpenDefaultKeyRings` for interpretation of the open flags.

Syntax

```
PGPError PGPOpenKeyRing(
             PGPContextRef pgpContext,
             PGPKeyRingOpenFlags openFlags,
             PGPFileSpecRef fileSpec,
             PGPKeySetRef *keySet );
```

Parameters

`pgpContext`	the target context
`openFlags`	the open option flags value
`fileSpec`	the target key ring file
`keySet`	the receiving field for the new key set

Flags

The open flags are interpreted as follows:

- `kPGPKeyRingOpenFlags_Create` – `TRUE` if the specified key ring file should be created if it doesn't already exist.
- `kPGPKeyRingOpenFlags_Mutable` – `TRUE` if the resultant key set should be made modifiable; `FALSE` if the resultant key set should be made read-only.
- `kPGPKeyRingOpenFlags_Trusted` – `TRUE` if any associated trust information should be included.
- `kPGPKeyRingOpenFlags_Private` – `TRUE` if the specified key ring file should be considered private; `FALSE` if the specified key ring file should be considered public.

Notes

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPReloadKeyRings

Forcibly re-establishes the key database associated with the specified key set from the key database source files.

Syntax

```
PGPError PGPReloadKeyRings ( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

Notes

The current implementation treats the target key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPCheckKeyRingSigs

Checks all signatures (or only those marked unchecked) of each key in the key database associated with the target key set. Each signature is assumed to exist in the key database associated with the look-up key set, which is typically all of the client's default keys.

Events of type `kPGPEvent_NullEvent` are sent during the course of processing, and the PGPsdk developer can choose to handle them with the optional event handler.

Syntax

```
PGPError PGPCheckKeyRingSigs(
             PGPKeySetRef keysToCheck,
             PGPKeySetRef keysSigning,
             PGPBoolean checkAll,
             PGPEventHandlerProcPtr eventHandler,
             PGPUserValue eventHandlerData );
```

Parameters

`keysToCheck`	the target key set
`keysSigning`	the look-up key set that contains the signing keys
`checkAll`	`TRUE` to check all signatures; `FALSE` to check only those marked as being unchecked
`eventHandler`	event handler or `NULL` to ignore any and all events (optional)
`eventHandlerData`	user-defined data to be passed to the event handler (optional)

Notes

This is a resource-intensive function, whose execution time can be quite lengthy.

The PGPsdk developer can choose to point the optional event handler to a function that implements a progress bar display, or anything else that the PGPsdk developer desires. `eventHandlerData` is passed to the event handler function, and has meaning only in conjunction with the event handler function (see the description for `kPGPEvent_NullEvent`).

The current implementation treats the target and look-up key sets as indirect parameters that reference key databases, rather than as explicit destinations and sources. Because of key filtering and the "live" nature of its resultant view-style key sets, the keys modified as a result any action by the optional event handler may be reflected in any key set based upon that key database, and further may or may not be reflected in the specified destination key set, depending upon its key filtering criteria.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPRevertKeyRingChanges

Undoes all changes made to the key database associated with the specified key set since it was last opened, or since it was last the target of a call to `PGPCommitKeyRingChanges`.

Syntax

```
PGPError PGPRevertKeyRingChanges( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

Notes

The current implementation treats the target key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function’s parameterization.

PGPCommitKeyRingChanges

Checks any signatures that are marked as unchecked, and re-propagates their trust model information and other attributes. It then writes any pending changes in the key database associated with the target key set to the backing store (disk or memory) upon which the key database is based.

Syntax

```
PGPError PGPCommitKeyRingChanges( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

Notes

Changes are only written to disk if and when the PGPsdk client calls this function.

The current implementation treats the target key set as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the “live” nature of its resultant view-style key sets, any keys modified by this function may be reflected in any key set based upon that key database, and further may or may not be reflected in the specified destination key set, depending upon its key filtering criteria.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function’s parameterization.

Key Set Management Functions

PGPNewKeySet

Creates a new memory-based *key database*, as well as an empty key set on that key database.

Syntax

```
PGPError PGPNewKeySet(
             PGPContextRef pgpContext,
             PGPKeySetRef *keySet );
```

Parameters

`pgpContext`	the target context
`keySet`	the receiving field for the new key set

Notes

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

The current implementation treats the resultant key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPNewEmptyKeySet

Creates a new, empty key set on the key database associated with the specified source key set.

Syntax

```
PGPError PGPNewEmptyKeySet(
             PGPKeySetRef baseKeySet,
             PGPKeySetRef *newKeySet );
```

Parameters

`baseKeySet`	the source key set
`newKeySet`	the receiving field for the new key set

Notes

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

The current implementation treats the supplied key set as an indirect parameter that references a key database, rather than as an explicit source.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPNewSingletonKeySet

Creates a key set that is *not associated with any key database*, and that contains only the specified seed key. This allows the PGPsdk developer to pass a single, specific key to a function that requires a key set argument.

Syntax

```
PGPError PGPNewSingletonKeySet (
             PGPKeyRef key, PGPKeySetRef *keySet );
```

Parameters

`key`	the source key
`keySet`	the receiving field for the new key set

Notes

This function does *not* create a new key database; the resultant key set contains only the one key.

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPUnionKeySets

Creates a new key set that is the union of the two source key sets

Syntax

```
PGPError PGPUnionKeySets(
             PGPKeySetRef firstKeySet,
             PGPKeySetRef secondKeySet,
             PGPKeySetRef *resultKeySet );
```

Parameters

`firstKeySet`	the first source key set
`secondKeySet`	the second source key set
`resultKeySet`	the receiving field for the new key set

Notes

The two source key sets *must* be in the same key database.

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPFreeKeySet

Decrements the reference count of the specified key set, and frees the key set if the reference count reaches zero.

Syntax

```
PGPError PGPFreeKeySet( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

PGPImportKeySet

Imports the specified keys from the specified input source in the options list into a new key set. By including an option that specifies sending null events, the PGPsdk developer can provide for tracking the progress of the function (see `PGPOSendNullEvents`).

Syntax

```
PGPError PGPImportKeySet(
             PGPContextRef pgpContext,
             PGPKeySetRef *keySet,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`keySet`	the receiving field for the resultant key set
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Import specific options include:

- `PGPOEventHandler`
- `PGPOInputBuffer`
- `PGPOInputFile`
- `PGPOInputFileFSSpec`
- `PGPOLocalEncoding`
- `PGPOSendNullEvents`

Notes

One of the following is required to specify the key source location:

- `PGPOInputBuffer`
- `PGPOInputFile`
- `PGPOInputFileFSSpec`

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPExportKeySet

Exports the specified keys in the specified key set to the output destination specified in the options list. By including an option that specifies sending null events, the PGPsdk developer can provide for tracking the progress of the function (see `PGPOSendNullEvents`).

Syntax

```
PGPError PGPExportKeySet(
             PGPKeySetRef keySet,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`keySet`	the target key set
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Export specific options include:

- `PGPOAllocatedOutputBuffer`
- `PGPOCommentString`
- `PGPODiscardOutput`
- `PGPOEventHandler`
- `PGPOExportPrivateKeys`
- `PGPOOutputBuffer`
- `PGPOOutputFile`
- `PGPOOutputFileFSSpec`
- `PGPOSendNullEvents`
- `PGPOVersionString`

Notes

One of the following is required to specify an output destination for functions that accept this option:

`PGPOAllocatedOutputBuffer`

`PGPOOutputBuffer`

`PGPOOutputFile`

`PGPOOutputFileFSSpec`

Exporting a key set and then importing it back in does *not* necessarily result in a key set that is identical to that initially exported. For example, if a key was signed as being non-exportable, then its signature data will be lost (see `PGPOExportable`).

PGPAddKeys

Copies all of the keys in the specified source key set to the key database associated with the specified destination (“to be augmented”) key set.

Syntax

```
PGPError PGPAddKeys(
            PGPKeySetRef keysToAdd,
            PGPKeySetRef keySet );
```

Parameters

`keysToAdd`	the source key set, which contains the keys to be added
`keySet`	the target (“to be augmented”) key set

Notes

The caller must call `PGPCommitKeyringChanges`.

The current implementation treats the destination key set as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the “live” nature of its resultant view-style key sets, the keys added by this function may appear in any key set based upon that key database

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function’s parameterization.

PGPRemoveKeys

Removes each of the keys in the specified source key set from the key database associated with the specified destination (“to be pruned”) key set.

Syntax

```
PGPError PGPRemoveKeys(
            PGPKeySetRef keysToRemove,
            PGPKeySetRef keySet );
```

Parameters

`keysToremove`	the source key set, which contains the keys to be removed
`keySet`	the target (“to be pruned”) key set

Notes

The current implementation treats the destination key set as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the “live” nature of its resultant view-style key sets, the keys removed by this function may disappear from any key set based upon that key database.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPPropagateTrust

Propagates the trust information across the key database associated with the specified key set.

Syntax

```
PGPError PGPPropagateTrust( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

Notes

The current implementation treats the destination key set as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the "live" nature of its resultant view-style key sets, the trust values propagated by this function may be reflected in any key set based upon that key database, and further may or may not be reflected in the specified destination key set, depending upon its key filtering criteria.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPCountKeys

Retrieves the number of keys in the specified key set.

Syntax

```
PGPError PGPCountKeys(
             PGPKeySetRef keySet, PGPUInt32 *numKeys );
```

Parameters

`keySet`	the target key set
`numKeys`	the receiving field for the key count

PGPKeySetIsMember

Returns `TRUE` if the specified key is in the specified key set.

Syntax

```
PGPBoolean PGPKeySetIsMember(
             PGPKeyRefkey, PGPKeySetRef keySet );
```

Parameters

`key`	the target key
`keySet`	the target key set

PGPKeySetIsMutable

Returns `TRUE` if the specified key set can be modified, that is if keys and their components (sub-keys, signatures, and user IDs) can be added to the key set, deleted from the key set, and have their properties changed in the key set.

Syntax

```
PGPBoolean PGPKeySetIsMutable( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

PGPKeySetNeedsCommit

Returns `TRUE` if there are any pending changes for the key database associated with the target key set.

Syntax

```
PGPBoolean PGPKeySetNeedsCommit( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

Notes

The current implementation treats the target key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

KeyFilter Functions

Filters are used to filter keys in a key set with `PGPFilterKeySet()`. Filters are also used when searching key servers to establish the search criteria.

PGPNewKeyBooleanFilter

Creates a filter which will match all keys on a given key Boolean property value.

Syntax

```
PGPError PGPNewKeyBooleanFilter(
             PGPContextRef pgpContext,
```

```
                    PGPKeyPropName property,
                    PGPBoolean match,
                    PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the Boolean property to examine
`match`	the Boolean value to match
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyCreationTimeFilter

Creates a key filter that will select those keys whose creation time meets the match criterion with respect to the specified creation time.

Syntax

```
PGPError PGPNewKeyCreationTimeFilter(
                    PGPContextRef pgpContext,
                    PGPTime creationTime,
                    PGPMatchCriterion match,
                    PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`creationTime`	the desired creation time value
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyExpirationTimeFilter

Creates a key filter that will select those keys whose expiration time meets the match criterion with respect to the specified expiration time.

Syntax

```
PGPError PGPNewKeyExpirationTimeFilter(
                    PGPContextRef pgpContext,
                    PGPTime expirationTime,
                    PGPMatchCriterion match,
```

```
                PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`expirationTime`	the desired expiration time value
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyDisabledFilter

Creates a key filter that will select for all disabled keys or for all enabled keys, depending on the value of the `disabled` argument.

Syntax

```
PGPError PGPNewKeyDisabledFilter(
                PGPContextRef pgpContext,
                PGPBoolean disabled,
                PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`disabled`	`TRUE` to match `disabled` keys; `FALSE` to match enabled keys
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyNumberFilter

Creates a filter which will match all keys on a given key numeric property value.

Syntax

```
PGPError PGPNewKeyNumberFilter(
                PGPContextRef pgpContext,
                PGPKeyPropName property,
                PGPUInt32 value,
                PGPMatchCriterion match,
                PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyTimeFilter

Creates a filter which will match all keys on a given key time property value.

Syntax

```
PGPError PGPNewKeyTimeFilter(
             PGPContextRef pgpContext,
             PGPKeyPropName property,
             PGPTime value,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold time value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyPropertyBufferFilter

Creates a filter which will match all keys on a given key binary data property value.

Syntax

```
PGPError PGPNewKeyPropertyBufferFilter(
             PGPContextRef context,
             PGPKeyPropName property,
             void *buffer,
             PGPSize length,
```

```
                    PGPMatchCriterion match,
                    PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`buffer`	the match threshold value buffer
`length`	the size (in bytes) of the buffer
`match`	how to match (=, !=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyRevokedFilter

Creates a key filter that will select for all revoked keys or for all non-revoked keys, depending on the value of the `revoked` argument.

Syntax

```
PGPError PGPNewKeyRevokedFilter(
                    PGPContextRef pgpContext,
                    PGPBoolean revoked,
                    PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`revoked`	TRUE to match **revoked** keys; FALSE to match non-revoked keys
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyEncryptAlgorithmFilter

Creates a key filter that will select those keys that use the specified public key algorithm.

Syntax

```
PGPError PGPNewKeyEncryptAlgorithmFilter(
                    PGPContextRef pgpContext,
                    PGPPublicKeyAlgorithm encryptAlgorithm,
                    PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`encryptAlgorithm`	the desired public key encryption algorithm
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

It may be useful to first determine if the desired public key encryption algorithm is available (see `PGPGetIndexedPublicKeyAlgorithmInfo`).

PGPNewKeyEncryptKeySizeFilter

Creates a key filter that will select those keys whose encryption key size (in bits) meets the match criterion with respect to the specified encryption key size.

Syntax

```
PGPError PGPNewKeyEncryptKeySizeFilter(
             PGPContextRef pgpContext,
             PGPUInt32 keySize,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`keySize`	the desired size of the encryption key (in bits)
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyFingerPrintFilter

Creates a key filter that will select for those keys having the specified fingerprint.

Syntax

```
PGPError PGPNewKeyFingerPrintFilter(
             PGPContextRef pgpContext,
             void const *fingerPrint,
             PGPSize fingerPrintLength,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`fingerPrint`	the desired key **fingerprint** in binary form
`fingerPrintLength`	the size of the desired fingerprint (in bytes)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeyIDFilter

Creates a key filter that will select for the specified key ID.

Syntax

```
PGPError PGPNewKeyIDFilter(
             PGPContextRef pgpContext,
             PGPKeyID const *keyID,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`keyID`	the desired key ID
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSubKeyBooleanFilter

Creates a filter which will match all keys on a given sub-key Boolean property value. Note that only the keys are filtered, not the matching subkeys.

Syntax

```
PGPError PGPNewSubKeyBooleanFilter(
             PGPContextRef pgpContext,
             PGPKeyPropName property,
             PGPBoolean match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the Boolean property to examine
`match`	the Boolean value to match
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSubKeyIDFilter

Creates a key filter that will select for the specified sub-key ID.

Syntax

```
PGPError PGPNewSubKeyIDFilter(
             PGPContextRef pgpContext,
             PGPKeyID const *subKeyID,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`subKeyID`	the desired sub-key ID
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSubKeyNumberFilter

Creates a filter which will match all keys on a given sub-key numeric property value. Note that only the keys are filtered, not the matching subkeys.

Syntax

```
PGPError PGPNewSubKeyNumberFilter(
             PGPContextRef pgpContext,
             PGPKeyPropName property,
             PGPUInt32 value,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSubKeyPropertyBufferFilter

Creates a filter which will match all keys on a given subkey binary data property value. Note that only the keys are filtered, not the matching subkeys.

Syntax

```
PGPError PGPNewSubKeyPropertyBufferFilter(
             PGPContextRef context,
             PGPKeyPropName property,
             void *buffer,
             PGPSize length,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`buffer`	the match threshold value buffer
`length`	the size (in bytes) of the buffer
`match`	how to match (=, !=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSubKeyTimeFilter

Creates a filter which will match all keys on a given sub-key time property value. Note that only the keys are filtered, not the matching subkeys.

Syntax

```
PGPError PGPNewSubKeyTimeFilter(
             PGPContextRef pgpContext,
```

```
PGPKeyPropName property,
PGPTime value,
PGPMatchCriterion match,
PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold time value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeySigAlgorithmFilter

Creates a key filter that will select those keys using the specified signature algorithm.

Syntax

```
PGPError PGPNewKeySigAlgorithmFilter(
             PGPContextRef pgpContext,
             PGPPublicKeyAlgorithm sigAlgorithm,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`sigAlgorithm`	the desired signature algorithm
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewKeySigKeySizeFilter

Creates a key filter that will select those keys whose signature key size (in bits) meets the match criterion with respect to the specified signature key size.

Syntax

```
PGPError PGPNewKeySigKeySizeFilter(
             PGPContextRef pgpContext,
             PGPUInt32 keySize,
             PGPMatchCriterion match,
```

```
                 PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`keySize`	the desired size of the signature key (in bits)
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSigBooleanFilter

Creates a filter which will match all keys on a given signature Boolean property value. Note that only the keys are filtered, not the matching signatures.

Syntax

```
PGPError PGPNewKeyBooleanFilter(
             PGPContextRef pgpContext,
             PGPKeyPropName property,
             PGPBoolean match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the Boolean property to examine
`match`	the Boolean value to match
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSigKeyIDFilter

Creates a key filter that will select those keys that were signed by the key having the specified key ID.

Syntax

```
PGPError PGPNewSigKeyIDFilter(
             PGPContextRef pgpContext,
             PGPKeyID const *keyID,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`keyID`	the desired signature key ID
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSigNumberFilter

Creates a filter which will match all keys on a given signature numeric property value. Note that only the keys are filtered, not the matching signatures.

Syntax

```
PGPError PGPNewSigNumberFilter(
             PGPContextRef pgpContext,
             PGPKeyPropName property,
             PGPUInt32 value,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSigPropertyBufferFilter

Creates a filter which will match all keys on a given signature binary data property value. Note that only the keys are filtered, not the matching signatures.

Syntax

```
PGPError PGPNewSigPropertyBufferFilter(
             PGPContextRef context,
             PGPKeyPropName property,
             void *buffer,
             PGPSize length,
             PGPMatchCriterion match,
```

```
                PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`buffer`	the match threshold value buffer
`length`	the size (in bytes) of the buffer
`match`	how to match (=, !=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewSigTimeFilter

Creates a filter which will match all keys on a given signature time property value. Note that only the keys are filtered, not the matching signatures.

Syntax

```
PGPError PGPNewSigTimeFilter(
                PGPContextRef pgpContext,
                PGPKeyPropName property,
                PGPTime value,
                PGPMatchCriterion match,
                PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold time value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewUserIDBooleanFilter

Creates a filter which will match all keys on a given user ID Boolean property value. Note that only the keys are filtered, not the matching user ID's.

Syntax

```
PGPError PGPNewUserIDBooleanFilter(
                PGPContextRef pgpContext,
```

```
                    PGPKeyPropName property,
                    PGPBoolean match,
                    PGPFilterRef *outFilter );
```

Parameters

pgpContext	the target context
property	name of the Boolean property to examine
match	the Boolean value to match
outFilter	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with PGPFreeFilter.

PGPNewUserIDNameFilter

Creates a key filter that will select keys whose user ID information matches the specified user name.

Syntax

```
PGPError PGPNewUserIDNameFilter(
                    PGPContextRef pgpContext,
                    char const *nameString,
                    PGPMatchCriterion match,
                    PGPFilterRef *outFilter );
```

Parameters

pgpContext	the target context
nameString	the desired user name
match	the match criterion
outFilter	the receiving field for the resultant key filter

Notes

Currently, the “name” component of a user ID is comprised of those characters up to, but not including, the first “<” character in the user ID.

The nameString argument length must not exceed kPGPMaxUserIDLength.

The caller is responsible for de-allocating the resultant key filter with PGPFreeFilter.

PGPNewUserIDNumberFilter

Creates a filter which will match all keys on a given user ID numeric property value. Note that only the keys are filtered, not the matching user ID’s.

Syntax

```
PGPError PGPNewUserIDNumberFilter(
```

```
PGPContextRef pgpContext,
PGPKeyPropName property,
PGPUInt32 value,
PGPMatchCriterion match,
PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`value`	the match threshold value
`match`	how to match (=, !=, <, >, <=, >=)
`outFilter`	the receiving field for the resultant key filter

Notes

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewUserIDStringBufferFilter

Creates a filter which will match all keys on a given user ID string property value. Note that only the keys are filtered, not the matching user ID's.

Syntax

```
PGPError PGPNewUserIDStringBufferFilter(
             PGPContextRef pgpContext,
             PGPUserIDPropName property,
             void *buffer,
             PGPSize length,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`property`	name of the property to examine
`buffer`	the match string buffer
`length`	the size (in bytes) of the buffer
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

This filter matches within the entire user ID string.

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewUserIDStringFilter

Creates a key filter that will select for keys whose user ID information matches the specified data string.

Syntax

```
PGPError PGPNewUserIDStringFilter(
             PGPContextRef pgpContext,
             char const *userIDString,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`userIDString`	the desired user ID
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

This filter matches within the entire user ID string.

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNewUserIDEmailFilter

Creates a key filter that will select for keys whose user ID information contains the specified email address.

Syntax

```
PGPError PGPNewUserIDEmailFilter(
             PGPContextRef pgpContext,
             char const *emailString,
             PGPMatchCriterion match,
             PGPFilterRef *outFilter );
```

Parameters

`pgpContext`	the target context
`emailString`	the desired user email address
`match`	the match criterion
`outFilter`	the receiving field for the resultant key filter

Notes

The “email” component of a user ID is comprised of those characters after the first “<“ character upto the first “>” character present.

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPNegateFilter

Creates a new key filter that will select those keys that the input key filter will exclude.

Syntax

```
PGPError PGPNegateFilter(
             PGPFilterRef filter,
             PGPFilterRef *outFilter );
```

Parameters

`filter`	the source key filter
`outFilter`	the receiving field for the resultant key filter

Notes

If the function returns an error, then the input filter is automatically freed. Otherwise, the input filter will be automatically freed when the resultant filter is freed. If the input filter should persist, then its reference count should be incremented with `PGPIncFilterRefCount`.

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPIntersectFilters

Creates a new key filter that is the logical intersection of the two input key filters. For example, for the resultant key filter to select an item, that item would have to be selectable by both of the input key filters.

Syntax

```
PGPError PGPIntersectFilters(
             PGPFilterRef filter1,
             PGPFilterRef filter2,
             PGPFilterRef *outFilter );
```

Parameters

`filter1`	the first source key filter
`filter2`	the second source key filter
`outFilter`	the receiving field for the resultant key filter

Notes

If the function returns an error, then the input filters are automatically freed. Otherwise, the input filters will be automatically freed when the resultant filter is freed. If the input filters should persist, then their reference counts should be incremented with `PGPIncFilterRefCount`.

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPUnionFilters

Creates a key filter that is the logical union of the two input key filters. For example, for the resultant key filter to select an item, that item would have to be selectable by either of the input key filters.

Syntax

```
PGPError PGPUnionFilters(
             PGPFilterRef filter1,
             PGPFilterRef filter2,
             PGPFilterRef *outFilter );
```

Parameters

`filter1`	first input key filter
`filter2`	second input key filter
`outFilter`	the receiving field for the resultant key filter

Notes

If the function returns an error, then the input filters are automatically freed. Otherwise, the input filters will be automatically freed when the resultant filter is freed. If the input filters should persist, then their reference counts should be incremented with `PGPIncFilterRefCount`.

The caller is responsible for de-allocating the resultant key filter with `PGPFreeFilter`.

PGPFreeFilter

Decrements the reference count of the specified key filter, and frees the key filter if the reference count reaches zero.

Syntax

```
PGPError PGPFreeFilter( PGPFilterReffilter );
```

Parameters

`filter`	the target key filter

PGPFilterKeySet

Applies the specified key filter to the specified key set. This yields a resultant key set that contains all of the keys from the source key set that meet the key filter criteria.

Syntax

```
PGPError PGPFilterKeySet(
             PGPKeySetRef origSet,
             PGPFilterRef filter,
             PGPKeySetRef *resultSet );
```

Parameters

`origSet`	the source key set
`filter`	the target key filter
`resultSet`	the receiving field for resultant key set

Notes

The resultant key set may be empty.

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPLDAPQueryFromFilter

Converts the key filter criteria to an LDAP key server format query string, which can then be passed to the key server for processing.

Syntax

```
PGPError PGPLDAPQueryFromFilter(
             PGPFilterRef filter, char **queryOut );
```

Parameters

`filter`	the target key filter
`queryOut`	the receiving field for a pointer to the resultant LDAP key server format query string

Notes

The caller is responsible for de-allocating the resultant query string with `PGPFreeData`.

Several key filter options are *not* supported by LDAP key servers (see Table 10-1).

PGPHKSQueryFromFilter

Converts the key filter criteria to an HTTP key server format query string, which can then be passed to the key server for processing.

Syntax

```
PGPError PGPHKSQueryFromFilter(
             PGPFilterRef filter, char **queryOut );
```

Parameters

`filter`	the target key filter
`queryOut`	the receiving field for a pointer to the resultant HTTP key server format query string

Notes

The caller is responsible for de-allocating the resultant query string with `PGPFreeData`.

A *significant* number of key filter options are *not* supported by HTTP key servers (see Table 10-1).

Key Iteration Functions

The PGPsdk supports both iterating through a key set and iterating through the sub-parts of an individual key. For iterating through a key set, the PGPsdk supports and requires the imposing of an ordering on that key set to yield a key list.

Whenever the iteration functions return `kPGPError_EndOfIteration`, the caller should treat the iterator's value as being undefined.

PGPOrderKeySet

Creates a key list from the target key set with the specified ordering, suitable for iteration (see this chapter's section on key iterator functions).

Syntax

```
PGPError PGPOrderKeySet(
             PGPKeySetRef keySet,
             PGPKeyOrdering order,
             PGPKeyListRef *keyList );
```

Parameters

`keySet`	the target key set
`order`	the ordering criteria, which recognizes `kPGP…Ordering` values
`keyList`	the receiving field for the resultant ordered key list

Notes

The PGPsdk supports only single-level ordering. For example, this function does not support creation of a key list ordered by expiration date within encryption key size.

The caller is responsible for de-allocating the resultant key list with `PGPFreeKeyList`.

PGPFreeKeyList

Decrements the reference count of the specified key list, and frees the key list if the reference count reaches zero.

Syntax

```
PGPError PGPFreeKeyList( PGPKeyListRef keySet );
```

Parameters

`keySet`	the target key list

PGPNewKeyIter

Creates an iterator on a list of keys. Note that a newly created iterator does not start out pointing at any particular key, user ID, or signature; in particular, it does not start out pointing at the first key in the key set. To access the first key with a newly-created iterator, you must first iterate to the 'next' item (for example, with `PGPKeyIterNext()`); to access any sub-part of the key, you must then futher iterate to the desired sub-part.

Syntax

```
PGPError PGPNewKeyIter(
             PGPKeyListRef keySet,
             PGPKeyIterRef *keyIter );
```

Parameters

`keySet`	the list of keys on which to iterate
`keyIter`	the receiving field for the iterator

Notes

A key list may have any number of iterators associated with it.

The caller is responsible for freeing the iterator with `PGPFreeKeyIter`.

PGPCopyKeyIter

Creates an exact copy of the source iterator, including its current index.

Syntax

```
PGPError PGPCopyKeyIter(
             PGPKeyIterRef iterOrig,
             PGPKeyIterRef *iterCopy );
```

Parameters

`iterOrig`	the source iterator
`iterCopy`	the receiving field for the copy of the iterator

Notes

The caller is responsible for de-allocating the resultant iterator copy with `PGPFreeKeyIter`.

PGPFreeKeyIter

Decrements the reference count of the specified iterator, and frees the iterator if the reference count reaches zero.

Syntax

```
PGPError PGPFreeKeyIter( PGPKeyIterRef iter );
```

Parameters

`iter`	the target iterator

PGPKeyIterIndex

Returns the current index value of the specified iterator.

Syntax

```
PGPInt32PGPKeyIterIndex( PGPKeyIterRef iter );
```

Parameters

`iter`	the target iterator

Notes

The caller should not infer anything based upon the returned index value.

PGPKeyIterKey

Yields the key associated with the current index value of the specified iterator.

Syntax

```
PGPError PGPKeyIterKey(
             PGPKeyIterRef iter, PGPKeyRef *key );
```

Parameters

`iter`	the target iterator
`key`	the receiving field for the resultant key

Notes

`kPGPError_EndOfIteration` is only returned if the key has been deleted.

PGPKeyIterSubKey

Yields the sub-key associated with the current index value of the specified iterator.

Syntax

```
PGPError PGPKeyIterSubKey(
             PGPKeyIterRef iter, PGPSubKeyRef*subKey );
```

Parameters

`iter`	the target iterator
`subKey`	the receiving field for the resultant `sub-`key

Notes

`kPGPError_EndOfIteration` is only returned if the sub-key has been deleted.

PGPKeyIterUserID

Yields the user ID associated with the current index value of the specified iterator.

Syntax

```
PGPError PGPKeyIterUserID(
             PGPKeyIterRef iter, PGPUserIDRef *userID );
```

Parameters

`iter`	the target iterator
`userID`	the receiving field for the resultant user ID

Notes

`kPGPError_EndOfIteration` is only returned if the user ID has been deleted.

PGPKeyIterSig

Yields the signature associated with the current index value of the specified iterator.

Syntax

```
PGPError PGPKeyIterSig(
             PGPKeyIterRef iter,PGPSigRef *sig );
```

Parameters

`iter`	the target iterator
`sig`	the receiving field for the resultant signature

Notes

`kPGPError_EndOfIteration` is only returned if the signature has been deleted.

PGPKeyIterMove

Moves the specified iterator by the specified relative number of keys, and yields the resultant key. Negative offsets move the iterator towards the beginning of the list; positive offsets move the iterator towards the end of the list.

Syntax

```
PGPError PGPKeyIterMove(
             PGPKeyIterRef iter,
             PGPInt32 relOffset,
             PGPKeyRef *key );
```

Parameters

`iter`	the target iterator
`relOffset`	the relative offset from the current position
`key`	the receiving field for the resultant key

Notes

If `kPGPError_EndOfIteration` is returned, then `key` will be set to `NULL`.

If `kPGPError_EndOfIteration` is returned, then the resultant key may have been deleted.

PGPKeyIterSeek

Scans the key set associated with the iterator, and returns the index (zero-based) of the first key that matches the specified search-for key.

Syntax

```
PGPInt32PGPKeyIterSeek(
             PGPKeyIterRef iter, PGPKeyRef key );
```

Parameters

`iter`	the target iterator
`key`	key to match

Notes

If the specified search-for key is not found, then the iterator is forcibly reset to point to the first key in the list. This should only happen if the search-for key was removed.

PGPKeyIterNext

Moves the specified iterator forward by one key, and yields the resultant key.

Syntax

```
PGPError PGPKeyIterNext(
             PGPKeyIterRef iter, PGPKeyRef *key );
```

Parameters

iter	the target iterator
key	the receiving field for the resultant key

Notes

This function is the equivalent of

```
PGPKeyIterMove( iter, 1, &key );
```

If kPGPError_EndOfIteration is returned, then key will be set to NULL.

If kPGPError_EndOfIteration is returned, then the resultant key may have been deleted.

PGPKeyIterNextSubKey

Moves the specified iterator forward by one subkey within the current key, and yields the resultant sub-key associated with the current key.

Syntax

```
PGPError PGPKeyIterNextSubKey(
             PGPKeyIterRef iter, PGPSubKeyRef *subKey );
```

Parameters

iter	the target iterator
subKey	the receiving field for the resultant sub-key

Notes

If kPGPError_EndOfIteration is returned, then subKey will be set to (PGPSubKeyRef *)NULL.

If kPGPError_EndOfIteration is returned, then the resultant sub-key may have been removed.

PGPKeyIterNextUserID

Moves the specified iterator forward by one user ID within the current key, and yields the resultant user ID associated with the current key.

Syntax

```
PGPError PGPKeyIterNextUserID(
             PGPKeyIterRef iter, PGPUserIDRef *userID );
```

Parameters

iter	the target iterator
userID	the receiving field for the resultant userID

Notes

If the current key has no associated user ID or the associated user ID has been removed, then the function returns kPGPError_BadParams.

If kPGPError_EndOfIteration is returned, then userID will be set to

`( PGPUserIDRef * )NULL`.

PGPKeyIterNextUIDSig

Moves the specified iterator forward by one user ID signature within the current user ID within the current key, and yields the resultant signature associated with the current user ID of the current key.

Syntax

```
PGPError PGPKeyIterNextUIDSig(
             PGPKeyIterRef iter, PGPSigRef *sig );
```

Parameters

`iter`	the target iterator
`sig`	the receiving field for the resultant signature

Notes

If the current key has no associated user ID or the associated user ID has been removed, then the function returns `kPGPError_BadParams`.

If `kPGPError_EndOfIteration` is returned, then `sig` will be set to `( PGPSigRef * )NULL`.

PGPKeyIterPrev

Moves the specified iterator backward by one key, and yields the resultant key.

Syntax

```
PGPError PGPKeyIterPrev(
             PGPKeyIterRef iter, PGPKeyRef *key );
```

Parameters

`iter`	the target iterator
`key`	the receiving field for the resultant key

Notes

This function is the equivalent of

```
PGPKeyIterMove( iter, -1, &key );
```

If `kPGPError_EndOfIteration` is returned, then `key` will be set to `NULL`. This may also indicate that what would have been the resultant key has been deleted.

PGPKeyIterPrevSubKey

Moves the specified iterator backward by one sub-key within the current key, and yields the resultant sub-key associated with the current key.

Syntax

```
PGPError PGPKeyIterPrevSubKey(
             PGPKeyIterRef iter, PGPSubKeyRef *key );
```

Parameters

`iter`	the target iterator
`key`	the receiving field for the resultant sub-key

Notes

A return value of `kPGPError_EndOfIteration` may also indicate that what would have been the resultant sub-key has been deleted.

PGPKeyIterPrevUserID

Moves the specified iterator backward by one user ID within the current key, and yields the resultant user ID associated with the current key.

Syntax

```
PGPError PGPKeyIterPrevUserID(
             PGPKeyIterRef iter, PGPUserIDRef *userID );
```

Parameters

`iter`	the target iterator
`userID`	the receiving field for the resultant user ID

Notes

If the current key has no associated user ID or the associated user ID has been removed, then the function returns `kPGPError_BadParams`.

If `kPGPError_EndOfIteration` is returned, then `userID` will be set to `NULL`.

PGPKeyIterPrevUIDSig

Moves the specified iterator backward by one user ID signature within the current user ID within the current key, and yields the resultant signature associated with the current user ID of the current key.

Syntax

```
PGPError PGPKeyIterPrevUIDSig(
             PGPKeyIterRef iter, PGPSigRef *sig );
```

Parameters

`iter`	the target iterator
`sig`	the receiving field for the resultant signature

Notes

If the current key has no associated user ID or the associated user ID has been removed, then the function returns `kPGPError_BadParams`.

If `kPGPError_EndOfIteration` is returned, then `sig` will be set to `NULL`.

PGPKeyIterRewind

Resets the iterator such that a subsequent `PGPKeyIterNextUserID` will yield the first user ID associated with the key.

Syntax

```
PGPError PGPKeyIterRewind( PGPKeyIterRef iter );
```

Parameters

`iter`	the target iterator

PGPKeyIterRewindSubKey

Resets the iterator such that a subsequent `PGPKeyIterNext` will yield the first key in the associated key list.

Syntax

```
PGPError PGPKeyIterRewindSubKey( PGPKeyIterRef iter );
```

Parameters

`iter`	the target iterator

PGPKeyIterRewindUserID

Resets the iterator such that a subsequent `PGPKeyIterNextUserID` will yield the first user ID associated with the key.

Syntax

```
PGPError PGPKeyIterRewindUserID( PGPKeyIterRef iter );
```

Parameters

`iter`	the target iterator

PGPKeyIterRewindUIDSig

Resets the iterator such that a subsequent `PGPKeyIterNextUIDSig` will yield the first signature associated with the current user ID of the current key.

Syntax

```
PGPError PGPKeyIterRewindUIDSig( PGPKeyIterRef iter );
```

Parameters

`iter`	the target iterator

Key reference count functions

The PGPsdk automatically tracks the number of data items pointing to a particular resource. For example, a given key set may be referenced by any number of key lists and / or key iterators. This not only results in a level of context independence, but also ensures that a resource’s memory is released only when its last reference is deleted. The PGPsdk also provides functions to support manual adjustment of a data item’s reference count.

PGPIncKeySetRefCount

Increments the reference count of the specified key set. This provides a mechanism for manually incrementing the reference count should it be necessary.

Syntax

```
PGPError PGPIncKeySetRefCount( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target key set

PGPIncFilterRefCount

Increments the reference count of the specified key filter. This provides a mechanism for manually incrementing the reference count should it be necessary.

Syntax

```
PGPError PGPIncFilterRefCount( PGPFilterRef filter );
```

Parameters

`filter`	the target key filter

PGPIncKeyListRefCount

Increments the reference count of the specified key list. This provides a mechanism for manually incrementing the reference count should it be necessary.

Syntax

```
PGPError PGPIncKeyListRefCount( PGPKeyListRef keySet );
```

Parameters

`keySet`	the target key list

Key manipulation functions

The key manipulation functions create, modify, and remove keys and their components (sub-keys, user ID's, signatures, and additional decryption keys). Since the parent item of a key or associated component must generally be active (not expired and not revoked) and mutable, most of the key manipulation functions can return one or more of the following error codes:

- `kPGPError_KeyExpired`
- `kPGPError_KeyRevoked`
- `kPGPError_ItemIsReadOnly`
- `kPGPError_ItemWasDeleted`

PGPGenerateKey

Generates a new key according to the specified options.

Syntax

```
PGPError PGPGenerateKey(
             PGPContextRef pgpContext,
             PGPKeyRef *key,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`key`	the receiving field for the generated key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to

terminate the argument list

Options

Key generation specific options include:

- `PGPOKeySetRef` (required)
- `PGPOKeyGenParams` (required)
- `PGPOKeyGenName` (required)
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPasskeyBuffer`
- `PGPOExpiration`
- `PGPOPreferredAlgorithms`
- `PGPOKeyGenFast`
- `PGPOAdditionalRecipientRequestKeySet`
- `PGPOCreationDate`
- `PGPOEventHandler`

Notes

Enough entropy must be available in the global random pool to generate the specified key type (see `PGPGetKeyEntropyRequired`).

Only one of `PGPOPassphrase`, `PGPOPassphraseBuffer` and `PGPOPasskeyBuffer` is allowed.

Key generation will fail with `PGPError_BadParams` if the specified key type cannot be used for signing.

The current implementation treats any destination key set specified with `PGPOKeySetRef` as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the “live” nature of its resultant view-style key sets, the key generated by this function may appear in any key set based upon that key database, and further may or may not appear in the specified destination key set, depending upon its key filtering criteria.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function’s parameterization.

PGPChangePassphrase

Changes the passphrase for the specified key.

Syntax

```
PGPError PGPChangePassphrase(
             PGPKeyRefkey,
             PGPOptionListRef firstOption,
             ...,
```

```
PGPOLastOption() );
```

Parameters

`key`	the target key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

The function expects two options - the first specifies the current passphrase, while the second specifies the new passphrase. The passphrases may be specified as any of the following:

- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPassKeyBuffer`

Notes

The specified key must be a private key, since public keys have no associated passphrase. Otherwise, the function returns `kPGPError_SecretKeyNotFound`.

If any sub-keys exist, then their passphrases should be changed via before changing the passphrase of their associated master key (see `PGPChangeSubKeyPassphrase()`).

PGPEnableKey

Marks a key as enabled for encryption and signing.

Syntax

```
PGPError PGPEnableKey( PGPKeyRef key );
```

Parameters

`key`	the target key

PGPDisableKey

Marks a key as disabled for encryption and signing. The target key is still enabled for decryption and verifying.

Syntax

```
PGPError PGPDisableKey( PGPKeyRef key );
```

Parameters

`key`	the target key

Notes

Axiomatically trusted keys cannot be disabled, and reflect `kPGPError_BadParams` (see `PGPUnsetKeyAxiomatic`).

PGPRevokeKey

Revokes the specified key according to the specified options.

```
PGPError PGPRevokeKey(
            PGPKeyRef key,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`key`	the key to be revoked
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Key revocation specific options include:

- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPassKeyBuffer`

Notes

In order to successfully revoke a key, its passphrase must be known. This implies that the function must be passed one of `PGPOPassphrase`, `PGPOPassphraseBuffer` and `PGPOPasskeyBuffer`.

If the specified key is already revoked and/or expired, then the function returns `kPGPError_NoErr`.

PGPSetKeyAxiomatic

Forces the specified private key to be axiomatically trusted. If `checkPassphrase` is `TRUE`, then any passphrase provided inthe option list must be both non-`NULL` and valid for the specified key (see `PGPUnsetKeyAxiomatic`). Upon successful return from this function the specified key will be enabled.

Syntax

```
PGPError PGPSetKeyAxiomatic(
            PGPKeyRef key,
            PGPBoolean checkPassphrase,
            char const *passphrase );
```

Parameters

`key`	the target key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Specific options include:

- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPassKeyBuffer`

Notes

The specified key must be a private key. Otherwise, the function returns `kPGPError_BadParams`.

Unless the key has just been created, a passphrase should be required to set such an unconditional trust level, but such a restriction is left to the PGPsdk developer and the needs of the application.

PGPUnsetKeyAxiomatic

Removes the axiomatic trust from the specified key (see `PGPSetKeyAxiomatic`).

Syntax

```
PGPError PGPUnsetKeyAxiomatic( PGPKeyRef key );
```

Parameters

`key`	the target key

Notes

If the specified key is already non-axiomatic, then the function returns `kPGPError_BadParams`.

PGPSetKeyTrust

Set the trust level of the specified key to that specified.

Syntax

```
PGPError PGPSetKeyTrust( PGPKeyRef key, PGPUInt32 trust );
```

Parameters

`key`	the target key
`trust`	the desired trust level

Notes

`kPGPKeyTrust_Undefined` and `kPGPKeyTrust_Ultimate` may not be used as `trust` argument values.

PGPCompareKeys

Compares the specified keys according to the specified ordering, and returns –`1`, `0`, or `1` depending on whether or not `key1` is less than, equal to, or greater than `key2`.

Syntax

```
PGPInt32 PGPCompareKeys(
             PGPKeyRef key1,
             PGPKeyRef key2,
             PGPKeyOrdering order );
```

Parameters

`key1`	the first target key
`key2`	the second target key
`order`	the ordering to be applied to the target keys, which recognizes `kPGP…Ordering` values

Notes

If the keys compare as equal with respect to the specified ordering, then the result reflects a comparison of their associated key IDs.

If both keys are found to be inactive, then the function returns 0 (zero).

PGPGenerateSubKey

Generates a new sub-key according to the specified options.

Syntax

```
PGPError PGPGenerateSubKey(
             PGPContextRef pgpContext,
             PGPSubKeyRef *subkey,
             PGPOptionListRef firstOption,
```

```
...,
PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`subkey`	the receiving field for the generated sub-key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Sub-key generation specific options include:

- `PGPOKeyGenMasterKey` (required)
- `PGPOKeyGenParams` (required)
- `PGPOPasskeyBuffer`
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPassKeyBuffer`
- `PGPOExpiration`
- `PGPOKeyGenFast`
- `PGPOCreationDate`
- `PGPOExportable`
- `PGPOFailBelowValidity`
- `PGPOHashAlgorithm`
- `PGPOInputBuffer`
- `PGPOKeySetRef`

Notes

Enough entropy must be available in the global random pool to generate the specified key type (see `PGPGetKeyEntropyRequired`).

The master key specified by the `PGPOKeyGenMasterKey` option must be active and mutable.

Only one of `PGPOPassphrase`, `PGPOPassphraseBuffer` and `PGPOPasskeyBuffer` is allowed.

Sub-key generation will fail with `PGPError_BadParams` if the specified key type cannot be used for signing.

Because of key filtering and the “live” nature of its resultant view-style key sets, the sub-key generated by this function may be reflected in any key set that contains its master key.

PGPRemoveSubKey

Removes the specified sub-key from its associated master key.

Syntax

```
PGPError PGPRemoveSubKey( PGPSubKeyRef subkey );
```

Parameters

`subkey`	the target sub-key

Notes

If the specified sub-key has already been removed from its associated master key, then the function returns `kPGPError_ItemWasDeleted`.

PGPChangeSubKeyPassphrase

Changes the passphrase for the specified sub-key according to the specified options.

Syntax

```
PGPError PGPChangeSubKeyPassphrase(
            PGPSubKeyRef subkey,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`subkey`	the target sub-key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Sub-key revocation specific options include:

- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPasskeyBuffer`

PGPRevokeSubKey

Revokes the specified sub-key according to the specified options.

```
PGPError PGPRevokeSubKey(
            PGPSubKeyRef subkey,
            PGPOptionListRef firstOption,
            ...,
```

```
                PGPOLastOption() );
```

Parameters

`subkey`	the target sub-key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Sub-key revocation specific options include:

- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPasskeyBuffer`

Notes

In order to successfully revoke a sub-key, its passphrase must be known. This implies that the function must be passed one of `PGPOPassphrase`, `PGPOPassphraseBuffer` and `PGPOPasskeyBuffer`.

The associated master key must be active and mutable.

If the specified sub-key has been removed from its associated master key, then the function returns `kPGPError_ItemWasDeleted`.

If the specified sub-key is already revoked and / or expired, then the function returns `kPGPError_NoErr`.

A return value of `kPGPError_SecretKeyNotFound` implies that the invoker is not authorized to revoke the specified sub-key.

PGPAddUserID

Creates an additional user ID for the specified key according to the specified options, and places it at the end of any existing list of user ID’s.

Syntax

```
PGPError PGPAddUserID(
                PGPKeyRef key,
                char const *name,
                PGPOptionListRef firstOption,
                ...,
                PGPOLastOption() );
```

Parameters

`key`	the key to which the user ID should be added
`name`	a character string (the user ID)
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

User ID specific options include:

- `PGPOPassphrase`
- `PGPOPassphraseBuffe`
- `PGPOPasskeyBuffer`

Notes

The `name` argument length must not exceed `kPGPMaxUserIDLength`.

Only one of `PGPOPassphrase`, `PGPOPassphraseBuffer` and `PGPOPasskeyBuffer` is allowed.

The specified key must be active and mutable.

A return value of `kPGPError_SecretKeyNotFound` implies that the invoker is not authorized to add user ID's to the specified key.

PGPRemoveUserID

Removes the specified user ID from its associated key.

Syntax

```
PGPError PGPRemoveUserID( PGPUserIDRef userID );
```

Parameters

`userID`	the target user ID

Notes

A return value of `kPGPError_BadParams` implies that the invoker attempted to remove the *only* user ID from the associated key, which is not allowed.

If the specified sub-key has already been removed from its associated key, then the function returns `kPGPError_ItemWasDeleted`.

PGPSetPrimaryUserID

Makes the specified user ID the primary user ID for its associated key.

Syntax

```
PGPError PGPSetPrimaryUserID( PGPUserIDRef userID );
```

Parameters

`userID`	the target user ID

Notes

The associated key must be active and mutable.

If the specified user ID has already been removed from its associated key, then the function returns `kPGPError_ItemWasDeleted`.

PGPCompareUserIDStrings

Compares the specified user ID strings, and returns –`1`, `0`, or `1` depending on whether or not `userIDString2` is less than, equal to, or greater than `userIDString1`.

Syntax

```
PGPInt32 PGPCompareUserIDStrings(
             char const *userIDString1,
             char const *userIDString2 );
```

Parameters

`userIDString1`	the first target user ID string
`userIDString2`	the second target user ID string

Notes

The `userIDString`*n* arguments length must not exceed `kPGPMaxUserIDLength`.

If the user ID strings compare as equal, then the result reflects a comparison of the associated key IDs.

If either `userIDString1` or `userIDString2` is `NULL`, then the function returns `0` (zero).

PGPSignUserID

Signs the key associated with the specified user ID with the specified signing key.

Syntax

```
PGPError PGPSignUserID(
             PGPUserIDRef userID,
             PGPKeyRef signingKey,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption ( void ) );
```

Parameters

`userID`	the target user ID
`signingKey`	the desired signing key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Signing specific options include:

- `PGPOExpiration`
- `PGPOExportable`
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOSigTrust`
- `PGPOSigRegularExpression`
- `PGPOCreationDate`
- `PGPOPasskeyBuffer`

Notes

Only one of `PGPOPassphrase`, `PGPOPassphraseBuffer` and `PGPOPasskeyBuffer` is allowed.

The associated key must be active and mutable.

If the specified user ID has been removed from its associated key, then the function returns `kPGPError_BadParams`, *not* `kPGPError_ItemWasDeleted`.

PGPRemoveSig

Removes the specified signature from its associated user ID of the associated key.

Syntax

```
PGPError PGPRemoveSig( PGPSigRef sig );
```

Parameters

`sig`	the signature to be removed

Notes

The associated key must be mutable.

If the specified signature has already been removed from its associated user ID, then the function returns `kPGPError_ItemWasDeleted`.

PGPRevokeSig

Revokes the specified signature from all keys in the key database associated with the specified target key set.

Syntax

```
PGPError PGPRevokeSig(
             PGPSigRef sig,
             PGPKeySetRef keySet,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`sig`	the target signature
`keySet`	the target key set
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Notes

If the specified signature has already been removed, then the function returns `kPGPError_ItemWasDeleted`; if it has been revoked, then the function returns `kPGPError_NoErr`.

The associated signing key must be active. If it does not exist, then the function returns `kPGPError_SecretKeyNotFound`.

The specified key set must be mutable.

The current implementation treats the destination key set as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the “live” nature of its resultant view-style key sets, the signature revocation resulting from this function may be reflected in any key set based upon that key database.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function’s parameterization.

PGPCountAdditionalRecipientRequests

Provides the number of **additional recipient request keys** that are available for the specified base key.

Syntax

```
PGPError PGPCountAdditionalRecipientRequests(
             PGPKeyRef baseKey, PGPUInt32 *numARRKeys );
```

Parameters

`baseKey`	the target key
`numARRKeys`	the receiving field for the resultant count

Notes

Use this count as the upper limit when indexing through the available additional recipient keys (see the sample code for `PGPGetIndexedAdditionalRecipientRequestKey`).

PGPGetIndexedAdditionalRecipientRequestKey

Provides a means of indexing through the available additional recipient request keys and retrieving each key, its key ID, and its class. All available additional recipient request keys are presumed to reside in the key database associated with the look-up key set.

Syntax

```
PGPError PGPGetIndexedAdditionalRecipientRequestKey(
             PGPKeyRef baseKey,
             PGPKeySetRef arrKeySet,
             PGPUInt32 index,
             PGPKeyRef *arrKey,
             PGPKeyID *arrKeyID,
             PGPByte *arrKeyClass );
```

Parameters

`baseKey`	the target key
`arrKeySet`	the look-up key set
`index`	the index (zero-based) of the desired additional recipient request key
`arrKey`	the receiving field for the n^{th} additional recipient request key
`arrKeyID`	the receiving field for the n^{th} additional recipient request key ID
`arrKeyClass`	the receiving field for the class of the additional recipient request key

Notes

The resultant key ID may not be used to access the additional recipient request key directly since key ID values are not unique.

One of `arrKeyID` and `arrKeyClass` may be `NULL` to indicate that the associated value should not be retrieved, but not both.

The class of the additional recipient request key is currently reserved for internal use, and the caller should not infer anything from its value.

The current implementation treats the look-up key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPGetSigCertifierKey

Searches the specified key set for the key associated with the specified signature.

Syntax

```
PGPError PGPGetSigCertifierKey(
             PGPSigRef sig,
             PGPKeySetRef allKeys,
             PGPKeyRef *sigKey );
```

Parameters

`sig`	the target signature
`allKeys`	the target key set
`sigKey`	the receiving field for the key associated with the target signature

PGPCountRevocationKeys

Provides the number of revocation keys that are available for the specified base key.

Syntax

```
PGPError PGPCountRevocationKeys(
             PGPKeyRef baseKey, PGPUInt32 *numRevKeys );
```

Parameters

`baseKey`	the target key
`numRevKeys`	the receiving field for the resultant count

Notes

Use this count as the upper limit when indexing through the available revocation keys (see the sample code for `PGPGetIndexedAdditionalRecipientRequestKey`).

PGPGetIndexedRevocationKey

Provides a means of indexing through the available revocation keys and retrieving each key, its key ID, and its class. All available revocation keys are presumed to reside in the key database associated with the look-up key set (see the sample code for `PGPGetIndexedAdditionalRecipientRequestKey`).

Syntax

```
PGPError PGPGetIndexedRevocationKey(
```

```
                    PGPKeyRef baseKey,
                    PGPKeySetRef revKeySet,
                    PGPUInt32 index,
                    PGPKeyRef *revKey,
                    PGPKeyID *revKeyID );
```

Parameters

`baseKey`	the target key
`arrKeySet`	the look-up key set
`index`	the index (zero-based) of the desired revocation key
`revKey`	the receiving field for the n^{th} revocation key
`revKeyID`	the receiving field for the n^{th} revocation key ID

Notes

The resultant key ID may not be used to access the revocation key directly since key ID values are not unique.

`arrKeyID` and `arrKeyClass` may be `NULL` to indicate that the associated value should not be retrieved.

The current implementation treats the look-up key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPPassphraseIsValid

Returns `TRUE` if the specified passphrase is valid for the specified key.

Syntax

```
PGPBoolean PGPPassphraseIsValid(
                    PGPKeySetRefkey,
                    const char *passphrase );
```

Parameters

`key`	the target key
`passphrase`	the assumed associated passphrase

Get property functions

PGPGetHashAlgUsed

Obtains the hash algorithm associated with the target key.

Syntax

```
PGPError PGPGetHashAlgUsed(
```

```
PGPKeyRef key, PGPHashAlgorithm *hashAlg );
```

Parameters

`key`	the target key
`hashAlg`	the receiving field for the hash algorithm value

PGPGetKeyBoolean

Retrieves the value of the specified boolean property of the specified key.

Syntax

```
PGPError PGPGetKeyBoolean(
            PGPKeyRefkey,
            PGPKeyPropName propName,
            PGPBoolean *propData );
```

Parameters

`key`	the target key
`propName`	the name of the target property, which recognizes `kPGPKeyProp`... values
`propData`	the receiving field for the target property value

Notes

If RSA encryption is not available, for example, an instance of the PGPsdk that supports only Elgamal encryption, then `propData` will be `FALSE` for both `kPGPKeyPropCanSign` and `kPGPKeyPropCanEncrypt`.

Example

```
PGPBoolean keyIsSecret;

err = PGPGetKeyBoolean( key,
    kPGPKeyPropIsSecret,
    &keyIsSecret );
if ( ( err == kPGPError_NoErr ) && ( keyIsSecret ) )
{
    /*
    ** Process secret key
    */
}
```

PGPGetKeyNumber

Retrieves the value of the specified numeric property of the specified key.

Syntax

```
PGPError PGPGetKeyNumber(
            PGPKeyRef key,
            PGPKeyPropName propName,
            PGPInt32 *propData );
```

Parameters

`key`	the target key
`propName`	the name of the desired property, which recognizes `kPGPKeyProp`... values
`propData`	the receiving field for the desired property value

PGPGetKeyPasskeyBuffer

Given the correct passphrase for a particular key, this function returns a buffer containing a corresponding binary "passkey". Passkeys can be used in most places in the PGPsdk in place of the passphrase, and this allows applications to keep a passphrase around in an uncompromised form. (PGP-brand products use this feature for caching passphrases over long periods of time.) For those PGPsdk functions that accept passphrase parameters, you can use the function `PGPOPasskeyBuffer()` to furnish a passkey buffer in place of a passphrase.

Syntax

```
PGPError PGPGetKeyPasskeyBuffer(
            PGPKeyRef key,
            void *passkeyBuffer,
            PGPOptionListRef firstOption,
            PGPOLastOption() );
```

Parameters

`key`	the target key
`passkeyBuffer`	the receiving buffer for the passkey
`firstOption`	the single option list instance
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Notes

When considering the size of your passkeyBuffer, note that the key property `kPGPKeyPropLockingBits` contains the number of bits (not bytes) needed to hold the passkey.

Options

The `firstOption` parameter must be either a `PGPOPassphrase()` or a `PGPOPassphraseBuffer()`, furnishing the passphrase for the indicated `key`.

PGPGetKeyPropertyBuffer

Retrieves the arbitrary binary data associated with the specified property of the specified key.

Syntax

```
PGPError PGPGetKeyPropertyBuffer(
```

```
            PGPKeyRef key,
            PGPKeyPropName propName,
            PGPSize availLength,
            void *propData,
            PGPSize *usedLength );
```

Parameters

`key`	the target key
`propName`	the name of the desired property, which recognizes `kPGPKeyProp`... values
`availLength`	the length of the receiving field for the desired property data
`propData`	the receiving field for the desired property data
`usedLength`	the receiving field for the resultant length of the desired property data

Notes

For a `propName` value of `kPGPPropPreferredAlgorithm`, a return value of `kPGPError_NoErr` with a resultant `usedLength` of zero indicates that no preferred algorithm is set.

PGPGetKeyTime

Retrieves the value of the specified date / time property of the specified key.

Syntax

```
PGPError PGPGetKeyTime(
            PGPKeyRef key,
            PGPKeyPropName propName,
            PGPTime *propData );
```

Parameters

`key`	the target key
`propName`	the name of the desired property, which recognizes `kPGPKeyProp`... values
`propData`	the receiving field for the desired property value

PGPGetSubKeyBoolean

Retrieves the value of the specified boolean property of the specified sub-key.

Syntax

```
PGPError PGPGetSubKeyBoolean(
            PGPSubKeyRef subkey,
            PGPKeyPropName propName,
            PGPBoolean *propData );
```

Parameters

`subkey`	the target sub-key
`propName`	the name of the desired property, which recognizes `kPGPKeyProp`... values
`propData`	the receiving field for the desired property data

Notes

Keys and sub-keys share the same `propName` values.

PGPGetSubKeyNumber

Retrieves the value of the specified numeric property of the specified sub-key.

Syntax

```
PGPError PGPGetSubKeyNumber(
             PGPSubKeyRef subkey,
             PGPKeyPropName propName,
             PGPInt32 *propData );
```

Parameters

`subkey`	the target sub-key
`propName`	which property you want to retrieve, which recognizes `kPGPKeyProp`... values
`propData`	the receiving field for the desired property

Notes

Keys and sub-keys share the same `propName` values.

PGPGetSubKeyPasskeyBuffer

Given the correct passphrase for a particular encryption sub-key, this function returns a buffer containing a corresponding binary “passkey”. Passkeys can be used in most places in the PGPsdk in place of the passphrase, and this allows applications to keep a passphrase around in an uncompromised form. (PGP-brand products use this feature for caching passphrases over long periods of time.) For those PGPsdk functions that accept passphrase parameters, you can use the function `PGPOPasskeyBuffer()` to furnish a passkey buffer in place of a passphrase.

Syntax

```
PGPError PGPGetSubKeyPasskeyBuffer(
             PGPSubKeyRef subKey,
             void *passkeyBuffer,
             PGPOptionListRef firstOption,
             PGPOLastOption() );
```

Parameters

`subKey`	the target sub-key
`passkeyBuffer`	the receiving buffer for the passkey
`firstOption`	the single option list instance
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Notes

When considering the size of your passkeyBuffer, note that the key property `kPGPKeyPropLockingBits` contains the number of bits (not bytes) needed to hold the passkey.

Options

The `firstOption` parameter must be either a `PGPOPassphrase()` or a `PGPOPassphraseBuffer()`, furnishing the passphrase for the indicated `subKey`.

PGPGetSubKeyPropertyBuffer

Retrieves the arbitrary binary data associated with the specified property of the specified sub-key.

Syntax

```
PGPError PGPGetSubKeyPropertyBuffer(
             PGPSubKeyRef subkey,
             PGPKeyPropName propName,
             PGPSize availLength,
             void *propData,
             PGPSize *usedLength );
```

Parameters

`subkey`	the target sub-key
`propName`	the name of the desired property, which recognizes `kPGPKeyProp`... values
`availLength`	the length of the receiving field for the desired property data
`propData`	the receiving field for the desired property data
`usedLength`	the receiving field for the resultant length of the desired property data

Notes

Keys and sub-keys share the same `propName` values.

For a `propName` value of `kPGPPropPreferredAlgorithm`, a return value of `kPGPError_NoErr` with a resultant `usedLength` of zero indicates that no preferred algorithm is set.

PGPGetSubKeyTime

Retrieves the value of the specified date / time property of the specified sub-key.

Syntax

```
PGPError PGPGetSubKeyTime(
             PGPSubKeyRef subkey,
             PGPKeyPropName propName,
             PGPTime *propData );
```

Parameters

`subkey`	the target sub-key
`propName`	the name of the desired property, which recognizes `kPGPKeyProp`... values
`propData`	the receiving field for the desired property value

Notes

Keys and sub-keys share the same `propName` values.

PGPGetUserIDBoolean

Retrieves the value of the specified boolean property of the specified user ID.

Syntax

```
PGPError PGPGetKeyBoolean(
             PGPUserIDRef userID,
             PGPUserIDPropName propName,
             PGPBoolean *propData );
```

Parameters

`userID`	the target user ID
`propName`	the name of the target property, which recognizes `kPGPUserIDProp`... values
`propData`	the receiving field for the target property value

PGPGetUserIDNumber

Retrieves the value of the specified numeric property of the specified user ID.

Syntax

```
PGPError PGPGetUserIDNumber(
             PGPUserIDRef userID,
             PGPUserIDPropName propName,
             PGPInt32 *propData );
```

Parameters

`userID`	the target user ID
`propName`	the name of the desired property, which recognizes `kPGPUserIDProp`... values
`propData`	the receiving field for the desired property value

Notes

Keys and sub-keys share the same `propName` values.

PGPGetUserIDStringBuffer

Retrieves the *C* language string associated with the specified property of the specified user ID.

Syntax

```
PGPError PGPGetUserIDStringBuffer(
             PGPUserIDRef userID,
             PGPUserIDPropName propName,
             PGPSize availLength,
             char *propString,
             PGPSize *usedLength );
```

Parameters

`userID`	the target user ID
`propName`	the name of the desired property, which recognizes `kPGPUserIDProp`... values
`availLength`	the length of the receiving field for the desired property data
`propString`	the receiving field for the desired property data
`usedLength`	the receiving field for the resultant length of the desired property data

Notes

`propString` should be a minimum of 256 bytes.

`usedLength` does *not* include the terminating `NUL`.

PGPGetSigBoolean

Retrieves the value of the specified boolean property of the specified signature.

Syntax

```
PGPError PGPGetSigBoolean(
             PGPSigRef sig,
             PGPSigPropName propName,
             PGPBoolean *propData );
```

Parameters

`sig`	the target signature
`propName`	the name of the desired property, which recognizes `kPGPSigProp`... values
`propData`	the receiving field for the desired property data

PGPGetSigNumber

Retrieves the value of the specified numeric property of the specified signature.

Syntax

```
PGPError PGPGetSigNumber(
             PGPSigRef sig,
             PGPSigPropName propName,
             PGPInt32 *propData );
```

Parameters

`sig`	the target signature
`propName`	the name of the desired property, which recognizes `kPGPSigProp`... values
`propData`	the receiving field for the desired property data

PGPGetSigPropertyBuffer

Retrieves the arbitrary binary data associated with the indicated signature.

Syntax

```
PGPError PGPGetSigPropertyBuffer(
             PGPSigRef sig,
             PGPKeyPropName propName,
             PGPSize bufferSize,
             void *propData,
             PGPSize *usedLength );
```

Parameters

`sig`	the target signature
`propName`	the name of the desired property, which recognizes `kPGPSigProp`... values
`bufferSize`	the length of the receiving field for the desired property data
`propData`	the receiving field for the desired property data
`usedLength`	the receiving field for the resultant length of the desired

property data

PGPGetSigTime

Retrieves the value of the specified date / time property of the specified signature.

Syntax

```
PGPError PGPGetSigTime(
            PGPSigRef sig,
            PGPSigPropName propName,
            PGPTime *propData );
```

Parameters

`sig`	the target signature
`propName`	the name of the desired property, which recognizes `kPGPSigProp`... values
`propData`	the receiving field for the desired property data

Convenience property functions

The "convenience property functions" encapsulate code that creates an iterator on the associated item, applies it to the specified key, outputs the associated property value, and frees the iterator.

PGPGetPrimaryUserID

Obtains the primary user ID of the specified key.

Syntax

```
PGPError PGPGetPrimaryUserID(
            PGPKeyRef key, PGPUserIDRef *userID );
```

Parameters

`key`	the target key
`userID`	the receiving field for the associated primary user ID

PGPGetPrimaryAttributeUserID

Returns the primary user ID designated for the indicated attribute type, for keys that have multiple attached attribute user ID's. To set this user ID, use `PGPSetPrimaryAttributeUserID()`.

Syntax

```
PGPError PGPGetPrimaryAttributeUserID(
            PGPKeyRef key,
```

```
PGPAttributeType attributeType,
PGPUserIDRef *outRef );
```

Parameters

`key`	the target key
`attributeType`	the desired attribute type
`outRef`	the receiving field for the associated primary user ID

PGPGetPrimaryUserIDNameBuffer

Retrieves the primary user ID name associated with the specified key, which is assumed to be a *C* language string.

Syntax

```
PGPError PGPGetPrimaryUserIDNameBuffer(
             PGPKeyRef key,
             PGPSize availLength,
             char *nameBuf,
             PGPSize *usedLength );
```

Parameters

`key`	the target key
`availLength`	the length of the receiving field for the associated primary user ID name
`nameBuf`	the receiving field for the associated primary user ID name
`usedLength`	the receiving field for the resultant length of the primary user ID name

Notes

`usedLength` does *not* include the terminating `NUL`.

PGPGetPrimaryUserIDValidity

Obtains the validity of the primary user ID associated with the specified key.

Syntax

```
PGPError PGPGetPrimaryUserIDValidity(
             PGPKeyRef key, PGPValidity *validity );
```

Parameters

`key`	the target key
`validity`	the receiving field for the validity value associated with the

user ID of the target key

Default Private Key Functions

PGPSetDefaultPrivateKey

Sets the default private key (nominally used for signing) to the specified key.

Syntax

```
PGPError PGPSetDefaultPrivateKey( PGPKeyRef key );
```

Parameters

key	the target key

Notes

The specified key must be active.

The specified key must be a **secret key** (kPGPKeyPropIsSecret), and must be able to sign (kPGPKeyPropCanSign). Otherwise, the functions returns kPGPError_BadParams.

The target key is forced to be axiomatically trusted (no passphrase is required).

PGPGetDefaultPrivateKey

Obtains the default private key, which is used for signing, for the key database associated with the specified key set.

Syntax

```
PGPError PGPGetDefaultPrivateKey(
            PGPKeySetRef keySet, PGPKeyRef *key );
```

Parameters

keySet	the target key set
key	the receiving field for the associated default private key

Notes

The current implementation treats the look-up key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

Key user-defined data functions

The PGPsdk provides the PGPsdk developer with a mechanism by which arbitrary data may be associated with keys and key elements. This data is of type `PGPUserValue`, and can be used for housekeeping, as pointers to data structures, or for any other user-defined purpose. When a key is first imported, all of these values are initialized to zero. These values are not saved with the key - they are only valid while the key or key element is in-memory.

PGPSetKeyUserVal

Associates a user-defined value or data structure with the specified key, provided that key is still in memory.

Syntax

```
PGPError PGPSetKeyUserVal(
            PGPKeyRef key, PGPUserValue userValue );
```

Parameters

`key`	the key with which the user value will be associated
`userValue`	the user-defined data

PGPSetSubKeyUserVal

Associates a user-defined value or data structure with the specified sub-key, provided that sub-key is still in memory.

Syntax

```
PGPError PGPSetSubKeyUserVal(
            PGPSubKeyRef subkey,
            PGPUserValue userValue );
```

Parameters

`subkey`	the sub-key with which the user value will be associated
`userValue`	the user-defined data

PGPSetSigUserVal

Associates a user-defined value or data structure with the specified signature, provided that signature is still in memory.

Syntax

```
PGPError PGPSetSigUserVal(
            PGPSigRef sig, PGPUserValue userValue );
```

Parameters

`sig`	the signature with which the user value will be associated
`userValue`	the user-defined data

PGPSetUserIDUserVal

Associates a user-defined value or data structure with the specified user ID, provided that user ID is still in memory.

Syntax

```
PGPError PGPSetUserIDUserVal(
            PGPUserIDRef userID,
            PGPUserValue userValue );
```

Parameters

`userID`	the user ID with which the user value will be associated
`userValue`	the user-defined data

PGPGetKeyUserVal

Obtains the user-defined data associated with the specified key (if any), and places it into the specified field.

Syntax

```
PGPError PGPGetKeyUserVal(
            PGPKeyRef key, PGPUserValue *userValue );
```

Parameters

`key`	the target key
`userValue`	the receiving field for the user-defined data

Notes

Any associated user data is always initialized to zeroes upon creation of a PGP data type instance.

PGPGetSubKeyUserVal

Obtains the user-defined data associated with the specified sub-key (if any), and places it into the specified field.

Syntax

```
PGPError PGPGetSubKeyUserVal(
            PGPSubKeyRef subkey,
            PGPUserValue *userValue );
```

Parameters

`subkey`	the target sub-key
`userValue`	the receiving field for the user-defined data

Notes

Any associated user data is always initialized to zeroes upon creation of a PGP data type instance.

PGPGetSigUserVal

Obtains the user-defined data associated with the specified signature (if any), and places it into the specified field.

Syntax

```
PGPError PGPGetSigUserVal(
            PGPSigRef sig, PGPUserValue *userValue );
```

Parameters

`sig`	the target signature
`userValue`	the receiving field for the user-defined data

Notes

Any associated user data is always initialized to zeroes upon creation of a PGP data type instance.

PGPGetUserIDUserVal

Obtains the user-defined data associated with the specified User ID (if any), and places it into the specified field.

Syntax

```
PGPError PGPGetUserIDUserVal(
            PGPUserIDRef userID,
            PGPUserValue *userValue );
```

Parameters

`userID`	the target user ID
`userValue`	the receiving field for the user-defined data

Notes

Any associated user data is always initialized to zeroes upon creation of a PGP data type instance.

KeyID functions

PGPImportKeyID

Imports the key ID.

Syntax

```
PGPError PGPImportKeyID(
             void const *data, PGPKeyID *keyID );
```

Parameters

`data`	the key ID data to import
`keyID`	the receiving field for the resultant key ID

Notes

`data` must be in the format produced by `PGPExportKeyID`, and must reference a buffer of at least `kPGPMaxExportedKeyIDSize` bytes in length

PGPExportKeyID

Exports the specified key ID.

Syntax

```
PGPError PGPExportKeyID(
             PGPKeyID const *keyID,
             PGPByte exportedData[
                 kPGPMaxExportedKeyIDSize ],
             PGPSize *exportedLength );
```

Parameters

`keyID`	the key ID to be exported
`exportedData`	the receiving field for the exported key ID data
`exportedLength`	the receiving field for the resultant length of the exported key ID data

PGPGetKeyIDString

Retrieves the string associated with the specified key ID.

Syntax

```
PGPError PGPGetKeyIDString(
             PGPKeyID const *keyID,
             PGPKeyIDStringType type,
             char outString[ kPGPMaxKeyIDStringSize ] );
```

Parameters

`keyID`	the target key ID
`type`	the type of key ID string to return, which recognizes `kPGPKeyIDString_`... values
`outString`	the receiving field for the associated key ID string

PGPGetKeyIDFromString

Creates a key ID corresponding to the specified key string.

Syntax

```
PGPError PGPGetKeyIDFromString(
             const char *string, PGPKeyID *keyID );
```

Parameters

`string`	the target string
`keyID`	the receiving field for the resultant key ID

Notes

The `string` argument length must not exceed `kPGPMaxKeyIDStringSize`.

PGPGetKeyByKeyID

Searches the key database associated with the specified key set for the key whose keyID and public key algorithm match those specified. This is especially useful for finding the keys of signing users, as well as any third party revocation keys or additional recipients (see `PGPGetKeyIDOfCertifier`, `PGPGetIndexedRevocationKey`, and `PGPGetIndexedAdditionalRecipientRequestKey`).

Syntax

```
PGPError PGPGetKeyByKeyID(
             PGPKeySetRef keySet,
             PGPKeyID const *keyID,
             PGPPublicKeyAlgorithm pubKeyAlgorithm,
             PGPKeyRef *key );
```

Parameters

`keySet`	the look-up key set
`keyID`	the target keyID
`pubKeyAlgorithm`	the public key algorithm used to generate the target keyID
`key`	the receiving field for the resultant key

Notes

Specifying the public key algorithm as `kPGPPublicKeyAlgorithm_Invalid` causes it to be ignored as a selection criteria.

The current implementation treats the look-up key set as an indirect parameter that references a key database, rather than as an explicit destination. Because of key filtering and the “live” nature of its resultant view-style key sets, the resultant key may or may not appear in the specified look-up key set, depending upon its key filtering criteria.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function’s parameterization.

PGPGetKeyIDFromKey

Creates a key ID corresponding to the specified key.

Syntax

```
PGPError PGPGetKeyIDFromKey(
             PGPKeyRef key, PGPKeyID *keyID );
```

Parameters

`key`	the target key
`keyID`	the receiving field for the resultant key ID

PGPGetKeyIDFromSubKey

Creates a key ID corresponding to the specified sub-key.

Syntax

```
PGPError PGPGetKeyIDFromSubKey(
             PGPSubKeyRef subkey, PGPKeyID *keyID );
```

Parameters

`subkey`	the target sub-key
`keyID`	the receiving field for the resultant key ID

PGPGetKeyIDOfCertifier

Retrieves the KeyID of the specified signature.

Syntax

```
PGPError PGPGetKeyIDOfCertifier(
             PGPSigRef sig, PGPKeyID *keyID );
```

Parameters

`sig`	the target signature
`keyID`	the receiving field for the associated KeyID

PGPCompareKeyIDs

Compares the key IDs, and returns `–1`, `0`, or `1` depending upon whether `keyID1` is less than `keyID2`, `keyID1` equals `keyID2`, or `keyID1` is greater than `keyID2`.

Syntax

```
PGPInt32 PGPCompareKeyIDs(
             PGPKeyID const *keyID1,
             PGPKeyID const *keyID2);
```

Parameters

`keyID1`	key ID
`keyID2`	key ID

Key Item Context Retrieval Functions

PGPGetKeySetContext

Returns the context associated with the specified key set.

Syntax

```
PGPContextRef PGPGetKeySetContext( PGPKeySetRef keySet );
```

Parameters

`keySet`	the target keySet

Notes

If the specified key set is invalid, then the returned context reference value is set to `kInvalidPGPContextRef`.

PGPGetKeyListContext

Returns the context associated with the specified key list.

Syntax

```
PGPContextRef PGPGetKeyListContext(
             PGPKeyListRef keyList );
```

Parameters

keyList	the target key list

Notes

If the specified key list is invalid, then the returned context reference value is set to `kInvalidPGPContextRef`.

PGPGetKeyIterContext

Returns the context associated with the specified key iterator.

Syntax

```
PGPContextRef PGPGetKeyIterContext(
             PGPKeyIterRef keyIter );
```

Parameters

keyIter	the target key iterator

Notes

If the specified key iterator is invalid, then the returned context reference value is set to `kInvalidPGPContextRef`.

PGPGetKeyContext

Returns the context associated with the specified key.

Syntax

```
PGPContextRef PGPGetKeyContext( PGPKeyRef key );
```

Parameters

key	the target key

Notes

If the specified key is invalid, then the returned context reference value is set to `kInvalidPGPContextRef`.

PGPGetSubKeyContext

Returns the context associated with the specified sub-key.

Syntax

```
PGPContextRef PGPGetSubKeyContext( PGPSubKeyRef subKey );
```

Parameters

subKey	the target sub-key

Notes

If the specified sub-key is invalid, then the returned context reference value is set to `kInvalidPGPContextRef`.

PGPGetUserIDContext

Returns the context associated with the specified user ID.

Syntax

```
PGPContextRef PGPGetUserIDContext( PGPUserIDRef userID );
```

Parameters

`userID`	the target user ID

Notes

If the specified user ID is invalid, then the returned context reference value is set to `kInvalidPGPContextRef`.

Key Share Functions

PGPSecretShareData

Divides a key into the specified number of shares, ensuring that each share is at least threshold bytes in length.

Syntax

```
PGPError PGPSecretShareData(
             PGPContextRef pgpContext,
             void const *inBuf,
             PGPSize inBufLength,
             PGPUInt32 threshold,
             PGPUInt32 numShares,
             void *outBuf );
```

Parameters

`pgpContext`	the target context
`inBuf`	the source key data
`inBufLength`	the size of the source share data (in bytes)
`threshold`	the minimum size (in bytes) of each share
`numShares`	the number of shares into which the source key data is to be divided
`outBuf`	the resultant share data

PGPSecretReconstructData

Syntax

```
PGPInt32 PGPSecretReconstructData(
             PGPContextRef pgpContext,
             void *inBuf,
```

```
                    PGPSize inBufLength,
                    PGPUInt32 numShares,
                    void *outBuf );
```

Parameters

`pgpContext`	the target context
`inBuf`	the source share data
`inBufLength`	the size of the source share data (in bytes)
`numShares`	the number of shares represented by the source share data
`outBuf`	the resultant share data

Misc. Key-related functions

PGPVerifyX509CertificateChain

Validates the first certificate in the specified chain by first looking in the specified chain, and then in the `rootCerts` chain to find a valid chain leading to a root key.

Both `certChain` and `rootCerts` are to be passed in the format that they appear in a TLS "server certificate" handshake message:

- 3 byte length for remainder

For each certificate:

- 3 byte certificate length
- certificate data

Syntax

```
PGPError PGPVerifyX509CertificateChain(
                    PGPContextRef pgpContext,
                    PGPByte *certChain,
                    PGPByte *rootCerts );
```

Parameters

`pgpContext`	the target context
`*certChain`	the target certificate chain
`*rootCerts`	a collection of trusted self-signed certificates

Notes

Returns `kPGPError_NoErr` if the certificate chain is found to be valid.

Option List Functions 3

Introduction

The PGPsdk provides a flexible and extensible mechanism for presenting arbitrary option specifications and data to functions accepting this mechanism.

Most of the option list management functions and the individual option functions use copy semantics. That is, they create their own copy of the arguments, and so allow the caller to delete the argument data upon return. This is very important in the case of passphrase and other sensitive data. In these cases, the caller should not only free the memory occupied by the argument, but also ensure that the memory is first erased. Additionally, the individual option functions allocate `PGPOptionListRef` instances that are automatically de-allocated once they are used in an option list management function, for example, `PGPBuildOptionList,` or as a sub-option, for example, `PGPOSignWithKey( …, PGPOPassphrase( … ), … )`.

The individual option functions do *not* return the data type `PGPError`; instead they always return the data type `PGOptionListRef`. However, an error may have occurred, and the resultant option list may not be valid (this is almost always due to `kPGPError_BadParams`, but may also be `kPGPError_OutOfMemory`). Since this condition can not be detected reliably until the resultant option list is actually used, the PGPsdk developer should always consider these option list functions as being a potential failure point for functions accepting option list arguments.

Header files

`pgpOptionList.h`

`pgpUserInterface.h`

Option list management functions

Option list management functions create and act upon persistent option lists, which must later be explicitly freed.

PGPNewOptionList

Creates an empty, persistent option list, which may then be the output target for subsequent `PGPAppendOptionList` and `PGPBuildOptionList` function calls.

Syntax

```
PGPError PGPNewOptionList(
             PGPContextRef pgpContext,
             PGPOptionListRef *outList );
```

Parameters

`pgpContext`	the target context
`outList`	the receiving field for the resultant option list

Notes

The caller is responsible for de-allocating the resultant option list via `PGPFreeOptionList`.

PGPBuildOptionList

Populates a persistent option list, replacing any previous content. Argument option list instances may be embedded option list function calls and/or previously built `PGPOptionListRef` instances, thus supporting modular assembly of option lists.

Syntax

```
PGPError PGPBuildOptionList(
             PGPContextRef pgpContext,
             PGPOptionListRef *outList,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`outList`	the receiving field for the resultant option list
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Notes

The caller is responsible for de-allocating the resultant option list via `PGPFreeOptionList`.

PGPCopyOptionList

Creates a persistent, exact copy of the source option list.

Syntax

```
PGPError PGPCopyOptionList(
             PGPOptionListRef optionListOrig,
             PGPOptionListRef *optionListCopy );
```

Parameters

`optionListOrig`	the source option list
`optionListCopy`	the receiving field for the copy of the option list

Notes

The caller is responsible for de-allocating the resultant copy of the option list via `PGPFreeOptionList`.

PGPAppendOptionList

Augments a persistent option list by appending the specified option(s) to any existing content. Argument option list instances may be embedded option list function calls and/or previously built `PGPOptionListRef` instances, thus supporting modular assembly of option lists.

Syntax

```
PGPError PGPAppendOptionList(
             PGPOptionListRef outList,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`outList`	the existing option list to which the specified option list instances will be appended
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

PGPFreeOptionList

Decrements the reference count of the specified option list and frees the option list if the reference count reaches zero.

Syntax

```
PGPError PGPFreeOptionList(
             PGPOptionListRef optionList );
```

Parameters

`optionList`	the existing option list to be de-allocated

Notes

Option lists that result from the inclusion of `PGPO`... functions in an argument list are automatically de-allocated upon return from the employing function. Such employing functions include, among others:

- `PGPEncode`
- `PGPDecode`
- `PGPBuildOptionList`
- `PGPAppendOptionList`
- `PGPAddJobOptionList`
- `PGPOUIDialogOptions`
- `PGPOUI...Dialog`

PGPAddJobOptions

Pass new option information to the job upon receipt of certain events. The job argument should be passed as `event->job`. Additional `PGPOptionListRef` arguments can be specified similarly to the way they are passed to `PGPEncode` and `PGPDecode`. However, only certain options can be set after each type of event.

Syntax

```
PGPError PGPAddJobOptions(
            PGPJobRef theJob,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`theJob`	the current job
`firstOption`	the initial option list instance
`...`	subsequent `option` list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Notes

`PGPAddJobOptions()` is found in `pgpEncode.h`.

Common Encode/Decode option list functions

The following functions are used to create `PGPOptionListRef` instances that specify the various common options to either `PGPDecode` or `PGPEncode.` These functions can be used as temporary inline arguments, or presented to `PGPAppendOptionList` and `PGPBuildOptionList` to augment or create existing persistent lists.

PGPOInputBuffer

Specifies that input is to be taken from the referenced buffer.

Syntax

```
PGPOptionListRef PGPOInputBuffer(
             PGPContextRef pgpContext,
             void const *inBuf,
             PGPSize inBufLength );
```

Parameters

`pgpContext`	the target context
`inBuf`	the desired input buffer
`inBufLength`	the length of the input data in the desired input buffer (in bytes)

Notes

One of `PGPOInputBuffer`, `PGPOInputFile`, and `PGPOInputFileFSSpec` is required to specify an input source for functions that accept this option.

If this option is specified in addition to an input file option, then the operation will fail with `kPGPError_BadParams`.

PGPOInputFile

Specifies that input is to be taken from the indicated file.

Syntax

```
PGPOptionListRef PGPOInputFile(
             PGPContextRef pgpContext,
             PGPFileSpecRef fileSpec );
```

Parameters

`pgpContext`	the target context
`fileSpec`	the desired input file specification

Notes

One of `PGPOInputBuffer`, `PGPOInputFile`, and `PGPOInputFileFSSpec` is required to specify an input source for functions that accept this option.

If this option is specified in addition to an input buffer option, then the operation will fail with `kPGPError_BadParams`.

PGPOInputFileFSSpec

(*MacOS platforms only*)

Specifies that input is to be taken from the indicated file, expressed as a Mac OS `FSSpec` record.

Syntax

```
PGPOptionListRef PGPOInputFileFSSpec(
             PGPContextRef pgpContext,
             const FSSpec *fileFSSpec );
```

Parameters

`pgpContext`	the target context
`fileFSSpec`	the FS specification of the desired input file

Notes

One of `PGPOInputBuffer`, `PGPOInputFile`, and `PGPOInputFileFSSpec` is required to specify an input source for functions that accept this option.

If this option is specified in addition to an input buffer option, then the operation will fail with `kPGPError_BadParams`.

PGPODiscardOutput

Specifies whether or not the output should be discarded, for example, sent to the null device.

Syntax

```
PGPOptionListRef PGPODiscardOutput(
             PGPContextRef pgpContext,
             PGPBoolean discardOutput );
```

Parameters

`pgpContext`	the target context
`discardOutput`	set to `TRUE` if the output is to be discarded

Notes

One of `PGPODiscardOutput`, `PGPOOutputFile`, `PGPOOutputBuffer`, and `PGPOOutputFileFSSpec` is required to specify an output destination for functions that accept this option.

If this option is specified with either an output file or an output buffer option, then the operation will fail with `kPGPError_BadParams`.

PGPOAllocatedOutputBuffer

Specifies that output should be placed in a dynamically allocated buffer. Upon completion of the operation, `outputBuf` will contain a pointer to the buffer, and `actualBufLength` will contain the length (in bytes) of the data placed into the output buffer.

Syntax

```
PGPOptionListRef PGPOAllocatedOutputBuffer(
             PGPContextRef pgpContext,
             void **outputBuf,
             PGPSize maximumBufLength,
             PGPSize *actualBufLength );
```

Parameters

`pgpContext`	the target context
`outputBuf`	the receiving field for a pointer to the allocated buffer
`maximumBufLength`	the maximum size to which the buffer may grow (in bytes)
`actualBufLength`	the receiving field for the actual size (in bytes) of the buffer

Notes

The caller is responsible for de-allocating the resultant buffer with `PGPFreeData.`

PGPOOutputBuffer

Specifies that output should be placed in a statically allocated buffer. Upon completion of the operation, outBufDataLength will contain the actual size (in bytes) of the output placed into the buffer.

Syntax

```
PGPOptionListRef PGPOOutputBuffer(
             PGPContextRef pgpContext,
             void *outBuf,
             PGPSize outBufLength,
             PGPSize *outBufDataLength );
```

Parameters

pgpContext	the target context
outBuf	the desired output buffer
outBufLength	the available size of the desired output buffer (in bytes)
outBufDataLength	the receiving field for the actual length (in bytes) of the data output

Notes

If outputDataLength is less than or equal to bufferLength, then all the output was successfully collected. If not, then some of the output data was lost.

One of PGPODiscardOutput, PGPOOutputFile, PGPOOutputBuffer, and PGPOOutputFileFSSpec is required to specify an output destination for functions that accept this option.

If this option is specified with either a discard output or an output file option, then the operation will fail with kPGPError_BadParams.

PGPOOutputFile

Specifies that output should be directed to the indicated file.

Syntax

```
PGPOptionListRef PGPOOutputFile(
             PGPContextRef pgpContext,
             PGPFileSpecRef fileSpec );
```

Parameters

pgpContext	the target context
fileSpec	the specification of the desired output file

Notes

One of `PGPODiscardOutput`, `PGPOOutputFile`, `PGPOOutputBuffer`, and `PGPOOutputFileFSSpec` is required to specify an output destination for functions that accept this option.

If this option is specified with either a discard output or an output buffer option, then the operation will fail with `kPGPError_BadParams`.

PGPOOutputFileFSSpec (*MacOS platforms only*)

Specifies that output should be directed to the indicated file, expressed as a Mac OS `FSSpec` record.

Syntax

```
PGPOptionListRef PGPOOutputFileFSSpec(
             PGPContextRef pgpContext,
             const FSSpec *fileFSSpec );
```

Parameters

`pgpContext`	the target context
`fileFSSpec`	the FS specification of the desired output file

Notes

One of `PGPODiscardOutput`, `PGPOOutputFile`, `PGPOOutputBuffer`, and `PGPOOutputFileFSSpec` is required to specify an output destination for functions that accept this option.

If this option is specified with either a discard output or an output buffer option, then the operation will fail with `kPGPError_BadParams`.

PGPOAppendOutput

Specifies whether or not output should be appended to any associated file or buffer, or should overwrite it.

Syntax

```
PGPOptionListRef PGPOAppendOutput(
             PGPContextRef pgpContext,
             PGPBoolean appendOutput );
```

Parameters

`pgpContext`	the target context
`appendOutput`	set to `TRUE` if the output is to be appended to any associated file or buffer

PGPOPGPMIMEEncoding

Specifies whether or not the output should be in MIME format. If `mimeEncoding` is `TRUE`, then `mimeBodyOffset` is initialized to zero, and `mimeSeparator` is initialized to an empty string, assuming that they are non-`NULL`.

Syntax

```
PGPOptionListRef PGPOPGPMIMEEncoding
             PGPContextRef pgpContext,
             PGPBoolean mimeEncoding,
             PGPSize *mimeBodyOffset,
             char mimeSeparator
             [   kPGPMimeSeparatorSize ] );
```

Parameters

`pgpContext`	the target context
`mimeEncoding`	set to `TRUE` if the output should be in **MIME** format
`mimeBodyOffset`	a field that will be used by the encoding process to hold the offset of the MIME body text, which is ignored if `mimeEncoding` is `FALSE`
`mimeSeparator`	a buffer that will be used by the encoding process to hold the MIME separator text, which must have a minimum length of `kPGPMimeSeparatorSize`, which is ignored if `mimeEncoding` is `FALSE`

Notes

This option forcibly sets `PGPOArmorOutput.`

PGPOOmitMIMEVersion

Specifies whether or not the MIME version should be included in the output, since some mailers automatically add the MIME version to their output. By specifying `TRUE`, the PGPsdk developer can avoid inclusion of two MIME version entries.

Syntax

```
PGPOptionListRef PGPOOmitMIMEVersion
             PGPContextRef pgpContext,
             PGPBoolean omitMIMEVersion );
```

Parameters

`pgpContext`	the target context
`omitMIMEVersion`	set to `TRUE` if the MIME version should *not* be included in the output

Notes

This option is only meaningful in conjunction with a `PGPOPGPMIMEEncoding` instance that enables MIME format.

PGPOLocalEncoding

Specifies the conditions under which the output should be converted to a platform-specific encoding. Currently, the PGPsdk only supports conversion to MacOS MacBinary format, and this function effectively does nothing on non-MacOS platforms.

Syntax

```
PGPOptionListRef PGPOLocalEncoding(
             PGPContextRef pgpContext,
             PGPLocalEncodingFlags localEncode );
```

Parameters

`pgpContext`	the target context
`localEncode`	the encoding to use

Flags

The local encoding flag values have the following meanings:

- `kPGPLocalEncoding_Auto` - effect conversion depending upon the output MacOS `OSType` file type
- `kPGPLocalEncoding_Force` - always effect conversion
- `kPGPLocalEncoding_NoMacBinCRCOkay` - flag the converted output such that a subsequent decode or signature verification ignores a failed CRC check
- `kPGPLocalEncoding_None` - no-op

The `kPGPLocalEncoding_Auto` and `kPGPLocalEncoding_Force` options are considered “main” options, and are mutually exclusive. `kPGPLocalEncoding_NoMacBinCRCOkay` and `kPGPLocalEncoding_None` are considered “modifier” options, and are intended to be `OR`’ed with one of the main options.

Notes

`kPGPLocalEncoding_NoMacBinCRCOkay` is primarily intended to provide compatibility with *PGP Version 2.6.2*.

When specified for `PGPDecode`, the option applies only to any detached signatures.

Generally, the PGPsdk developer should always specify `kPGPLocalEncoding_Force` since this:

- ensures that no data will be lost
- is ignored for output on non-MacOS platforms
- is recognized for input by versions 5.5 and later of PGP software products on non-MacOS platforms

Example

```
tOptListRef = PGPOLocalEncoding(
        pgpContext,
        ( kPGPLocalEncoding_Force |
          kPGPLocalEncoding_NoMacBinCRCOkay ) );
```

PGPOOutputLineEndType

Specifies the type of line endings to use when generating text output.

Syntax

```
PGPOptionListRef PGPOOutputLineEndType(
            PGPContextRef pgpContext,
            PGPLineEndType lineEndType );
```

Parameters

`pgpContext`	the target context
`lineEndType`	the line ending to use

Notes

This option is only meaningful in conjunction with `PGPOArmorOutput`. If this option is not supplied, then the default line ending for the local platform is used.

PGPODetachedSig

For `PGPEncode`, creates a detached signature for the message. No sub-options are defined at this time.

For `PGPDecode`, specifies the input source to be used to verify any associated detached signature. In this case, one of `PGPOInputBuffer`, `PGPOInputFile`, and `PGPOInputFileFSSpec` is required.

Syntax

```
PGPOptionListRef PGPODetachedSig(
             PGPContextRef pgpContext,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Detached signature specific options include:

- `PGPOInputBuffer`
- `PGPOInputFile`
- `PGPOInputFileFSSpec`

Common encrypting and signing option list functions

PGPOConventionalEncrypt

Conventionally encrypt the message.

Syntax

```
PGPOptionListRef PGPOConventionalEncrypt(
            PGPContextRef pgpContext,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Conventional encryption specific options include:

- `PGPOPassphrase`
- `PGPOPassphraseBuffer`

Notes

This option requires a `PGPOPassphrase` sub-option to specify the conventional encryption key.

PGPOCipherAlgorithm

Specifies the algorithm to use for encryption. This is currently meaningful only in conjunction with conventional encryption; otherwise the choice of encryption algorithm is based on the encrypt-to keys.

Syntax

```
PGPOptionListRef PGPOCipherAlgorithm(
            PGPContextRef pgpContext,
            PGPCipherAlgorithm algID );
```

Parameters

`pgpContext`	the target context
`algID`	the cipher algorithm to use

PGPOEncryptToKey

Encrypt the plain text to the specified key.

Syntax

```
PGPOptionListRef PGPOEncryptToKey(
             PGPContextRef pgpContext,
             PGPKeyRef keyRef );
```

Parameters

`pgpContext`	the target context
`keyRef`	the target key

Notes

To encrypt the plain text with multiple keys, include an instance of this option in the `PGPEncode` option list for each key. There is no preset limit to the number of instances.

If the number of individual encrypt-to keys is large or if multiple data instances are to be encrypted, then it may be simpler to collect the keys as a key set and use `PGPOEncryptToKeySet`.

PGPOEncryptToKeySet

Encrypt the plain text to each key in the key set. This option may be used multiple times in one call.

Syntax

```
PGPOptionListRef PGPOEncryptToKeySet(
             PGPContextRef pgpContext,
             PGPKeySetRef keySet );
```

Parameters

`pgpContext`	the target context
`keySet`	the target key set

Notes

To encrypt the plain text to each key in multiple key sets, include an instance of this option in the `PGPEncode` option list for each key set. There is no preset limit to the number of instances.

PGPOEncryptToUserID

Encrypt the plain text to the key associated with the specified user ID.

Syntax

```
PGPOptionListRef PGPOEncryptToUserID(
             PGPContextRef pgpContext,
             PGPUserIDRef userIDRef );
```

Parameters

`pgpContext`	the target context
`userIDRef`	the target user ID

Notes

To encrypt the plain text with the keys associated with multiple user IDs, include an instance of this option in the `PGPEncode` option list for each user ID. There is no preset limit to the number of instances.

This function is believed to be of limited use, and may not be supported in future versions of the PGPsdk.

PGPOHashAlgorithm

Use the specified algorithm as the hash algorithm for signatures. For example, force the use of the SHA-1 algorithm in an RSA signature.

Syntax

```
PGPOptionListRef PGPOHashAlgorithm(
             PGPContextRef pgpContext,
             PGPHashAlgorithm algID );
```

Parameters

`pgpContext`	the target context
`algID`	the desired hash algorithm

Notes

DSS keys unconditionally use the SHA-1 algorithm, and are unaffected by this option.

PGPOSignWithKey

Sign the message or file with the specified key.

Syntax

```
PGPOptionListRef PGPOSignWithKey(
            PGPContextRef pgpContext,
            PGPKeyRef sigKey,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`sigKey`	the desired signing key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Signing specific options include:

- `PGPOPasskeyBuffer`
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`

Notes

Any required passphrase should be specified with a sub-option. A passphrase event is posted if all of the following conditions exist:

- no passphrase sub-option is specified
- the target key requires a passphrase
- an event handler is defined (see `PGPOEventHandler`)

PGPOWarnBelowValidity

For encryption and signature verification, specifies that a warning event be sent for any encryption or signing key having a validity level less than that specified.

Syntax

```
PGPOptionListRef PGPOWarnBelowValidity(
            PGPContextRef pgpContext,
            PGPValidity minValidity );
```

Parameters

`pgpContext`	the target context
`minValidity`	the desired validity threshold

PGPOFailBelowValidity

For encryption, specifies that a fatal error be recognized for an encryption key having a validity level less than that specified. For signature verification, specifies that the generated signature event `keyValidity` member be set to `kPGPValidity_Invalid`.

Syntax

```
PGPOptionListRef PGPOFailBelowValidity(
            PGPContextRef pgpContext,
            PGPValidity minValidity );
```

Parameters

`pgpContext`	the target context
`minValidity`	the desired validity threshold

Encode-only Option List Functions

PGPOAskUserForEntropy

Specifies whether or not the user should be prompted to provide additional entropy if the global random pool entropy level drops below its minimum.

Syntax

```
PGPOptionListRef PGPOAskUserForEntropy(
            PGPContextRef pgpContext,
            PGPBoolean askUser );
```

Parameters

`pgpContext`	the target context
`askUser`	set to `TRUE` if the user should be prompted for additional entropy

Notes

If the user is not to be prompted and the entropy drops below minimum, then the operation will fail with `kPGPError_OutOfEntropy`.

PGPODataIsASCII

Force all line endings to <CR><LF> pairs prior to encoding or signing. This flags the cipher text such that `PGPDecrypt` will generate the plain text with output line endings appropriate to the decoding platform.

Syntax

```
PGPOptionListRef PGPODataIsASCII(
            PGPContextRef pgpContext,
            PGPBoolean dataIsASCII );
```

Parameters

`pgpContext`	the target context
`dataIsASCII`	set to `TRUE` if the input data should be interpreted as ASCII

PGPORawPGPInput

Indicates whether or not the input is already in binary PGP format. This simplifies decryption of messages that are multiply signed and/or multiply encrypted.

Syntax

```
PGPOptionListRef PGPORawPGPInput(
            PGPContextRef pgpContext,
            PGPBoolean isRawPGPInput );
```

Parameters

`pgpContext`	the target context
`isRawPGPInput`	set to `TRUE` if the input is assumed to be in raw PGP format

Notes

`PGPORawPGPInput` is intended for internal use by the PGPsdk routines, and client code should specify this option rarely, if ever.

PGPOForYourEyesOnly

Encrypt in "for your eyes only" mode. This flags the cipher text such that the output events generated during decoding will reflect `TRUE` for the `forYourEyesOnly` member of the `PGPEventOutputData`. This in turn alerts the client to the fact that the resultant plain text should not be saved to disk, or otherwise made available to other recipients.

Syntax

```
PGPOptionListRef PGPOForYourEyesOnly(
            PGPContextRef pgpContext,
            PGPBoolean forYourEyesOnly );
```

Parameters

`pgpContext`	the target context
`forYourEyesOnly`	set to `TRUE` to enable "for your eyes only" encryption mode

Notes

This option is not enforceable by the encrypting client - the decrypting client may always choose to ignore events entirely or simply ignore this indicator.

PGPOArmorOutput

Ensures that all output is encoded as 7-bit ASCII. For example, a 32-bit binary numeric value of `688,798,386` would be rendered as the ASCII text string `“290E3AB2”`, assuming big-endian encoding.

Syntax

```
PGPOptionListRef PGPOArmorOutput(
             PGPContextRef pgpContext,
             PGPBoolean armorOutput );
```

Parameters

`pgpContext`	the target context
`armorOutput`	set to `TRUE` if the resultant output should be ASCII encoded

PGPOFileNameString

Sets the ‘suggested’ name for the decrypted file, which is stored within the encrypted file. By default, the internal file name string is set to the name of the input file.

For example, suppose we encrypt a file called “Profits.xls”, naming the encryption output file “Secret.pgp”. If the internal ‘suggested’ filename string is set to “Profits.xls”, then upon decryption the unencoded file will also be named “Profits.xls”.

Syntax

```
PGPOptionListRef PGPOFileNameString(
             PGPContextRef pgpContext,
             char const *fileNameString );
```

Parameters

`pgpContext`	the target context
`fileNameString`	the suggested filename for the decrypted file,

expressed as a null-terminated C string.

PGPOClearSign

Clear-sign the message, that is, output the text as lexical sections with the appropriate PGP delimiters, but do not encrypt the plain text. In this way, messages can be sent "in the clear" while still providing for authentication. This option forcibly sets both `PGPOArmorOutput` and `PGPODataIsASCII`.

Syntax

```
PGPOptionListRef PGPOClearSign(
             PGPContextRef pgpContext,
             PGPBoolean clearSign );
```

Parameters

`pgpContext`	the target context
`clearSign`	set to TRUE if the resultant output should be clear-signed

Decode-only Option List Functions

PGPOImportKeysTo

If any keys are found in the input, add them to the specified key set.

Syntax

```
PGPOptionListRef PGPOImportKeysTo(
             PGPContextRef pgpContext,
             PGPKeySetRef keySet );
```

Parameters

`pgpContext`	the target context
`keySet`	the target key set

PGPOPassThroughIfUnrecognized

Indicate whether or not unrecognized lexical sections should post an error.

Syntax

```
PGPOptionListRef PGPOPassThroughIfUnrecognized(
             PGPContextRef pgpContext,
             PGPBoolean passThrough );
```

Parameters

`pgpContext`	the target context
`passThrough`	set to TRUE if unrecognized lexical sections should *not* post an

error

PGPOPassThroughClearSigned

Option for `PGPDecode()` to request that clear-signed data appear at the output of the operation with the signature data intact. The default behavior for `PGPDecode()` is to remove wrapping signature information.

Syntax

```
PGPOptionListRef PGPOPassThroughClearSigned(
            PGPContextRef context,
            PGPBoolean passThrough );
```

Parameters

`pgpContext`	the target context
`passThrough`	set to `TRUE` to enable passthrough of clear-signed data

PGPOPassThroughKeys

Option for `PGPDecode()` to request that embedded key blocks appear at the output of the operation. The default behavior for `PGPDecode()` is to remove embedded key blocks, and to import the keys into a key set if an `PGPOImportKeysTo()` option is used.

Syntax

```
PGPOptionListRef PGPOPassThroughKeys(
            PGPContextRef context,
            PGPBoolean passThrough );
```

Parameters

`pgpContext`	the target context
`passThrough`	set to `TRUE` to enable passthrough of keys

PGPOSendEventIfKeyFound

Enable or disable sending `kPGPEvent_KeyFound` events, which allows an event handler to decide what to do with keys in the input.

Syntax

```
PGPOptionListRef PGPOSendEventIfKeyFound(
            PGPContextRef pgpContext,
            PGPBoolean sendEventIfKeyFound );
```

Parameters

`pgpContext`	the target context
`sendEventIfKeyFound`	set to `TRUE` to enable sending of `kPGPEvent_KeyFound events`

PGPORecursivelyDecode

Option for `PGPDecode()` to tell the SDK to check the decrypted message for any clear-signed information, and then verify that information. This check takes place after decryption. This functionality is intended to accomodate cases in which a clear-signed message is subsequently encrypted in a separate, explicit encryption operation (as opposed to performing an encrypt-and-sign, which is always regarded as a single operation by the SDK).

Syntax

```
PGPOptionListRef PGPOSendEventIfKeyFound(
             PGPContextRef pgpContext,
             PGPBoolean recurse );
```

Parameters

`pgpContext`	the target context
`recurse`	set to `TRUE` to enable recursive decoding

(Sub-)Key Generation, Augmentation, and Revocation Option List Functions

The following functions are used to create `PGPOptionListRef` instances that specify the various common options to `PGPGenerateKey`, `PGPGenerateSubKey`, `PGPGetKeyEntropyNeeded`, `PGPAddUserID`, and `PGPSignUserID`. These functions can be used as temporary inline arguments, or used with `PGPAppendOptionList` and `PGPBuildOptionList` to augment or create existing persistent lists.

PGPOAdditionalRecipientRequestKeySet

Establish the specified key(s) as additional recipient request key(s) when generating keys with `PGPGenerateKey.`

Syntax

```
PGPOptionListRef PGPOAdditionalRecipientRequestKeySet(
             PGPContextRef pgpContext,
             PGPKeySetRef arrKeySet,
             PGPByte arrKeyClass );
```

Parameters

`pgpContext`	the target context
`arrKeySet`	the key set containing the additional recipient request keys
`arrKeyClass`	the class of the additional recipient request keys

Notes

This option is valid for `PGPGenerateKey` only.

`arrKeyClass` is currently ignored, and should be specified as `( PGPByte )0`.

PGPOKeyGenName

Establish the name to be used when generating keys with `PGPGenerateKey`.

Syntax

```
PGPOptionListRef PGPOKeyGenName(
             PGPContextRef pgpContext,
             const void *name,
             PGPSize nameLength );
```

Parameters

`pgpContext`	the target context
`name`	the desired name
`nameLength`	the length (in bytes) of the desired name, which must be between 1 (one) and 256

Notes

This option is valid for `PGPGenerateKey only`.

PGPOKeyGenMasterKey

Specifies the key on which a sub-key will be generated.

Syntax

```
PGPOptionListRef PGPOKeyGenMasterKey(
             PGPContextRef pgpContext,
             PGPKeyRef masterKey );
```

Parameters

`pgpContext`	the target context
`masterKey`	the “parent” key

Notes

This option is valid for `PGPGenerateSubKey` `only`.

PGPOExportPrivateKeys

Indicate whether or not private keys should be included when exporting key sets.

Syntax

```
PGPOptionListRef PGPOExportPrivateKeys(
             PGPContextRef pgpContext,
             PGPBoolean exportPrivateKeys );
```

Parameters

`pgpContext`	the target context
`exportPrivateKeys`	set to TRUE to include private keys in exported key sets

PGPOKeyGenFast

Indicate whether or not keys should be generated in “fast” mode, that is, based on “known” primes instead of dynamically generated primes.

Syntax

```
PGPOptionListRef PGPOKeyGenFast(
             PGPContextRef pgpContext,
             PGPBoolean fastGen );
```

Parameters

`pgpContext`	the target context
`fastGen`	set to TRUE to enable “fast” key generation mode

PGPOKeyGenParams

Establishes the public key algorithm and key size (in bits) to be used when generating keys or sub-keys, as well as when determining the entropy required to generate such keys or sub-keys.

Syntax

```
PGPOptionListRef PGPOKeyGenParams(
             PGPContextRef pgpContext,
             PGPPublicKeyAlgorithm pubKeyAlg,
             PGPUInt32 keySize );
```

Parameters

`pgpContext`	the target context
`pubKeyAlg`	the desired public key algorithm
`keySize`	the desired key size (in bits), which must be at least 512

Notes

The permissible key size values depend upon the choice of algorithm.

This option is required by those functions that accept it.

PGPOCreationDate

Sets the creation date of keys, sub-keys, and signatures generated for the specified context. When a key, sub-keys, or signature is actually generated, the PGPsdk sets it's creation date to that specified.

Syntax

```
PGPOptionListRef PGPOCreationDate(
             PGPContextRef pgpContext,
             PGPTime creationDate );
```

Parameters

`pgpContext`	the target context
`creationDate`	the desired creation date, expressed as a `PGPTime` value

Notes

If this option is not supplied, then the creation date defaults to "now".

Use the PGPsdk utility function `PGPGetPGPTimeFromStdTime` to convert a Standard *C* Library `time_t` value to a `PGPTime` value.

Since the system's time-of-day clock can be manually set to any date or time, there are no restrictions on the specified date being in the past or in the future. However, the creation date *must* be before any specified expiration date (see `PGPOExpiration`).

PGPOExpiration

Sets the expiration date of keys and their component items generated for the specified context. Whenever a key or component is actually generated, the PGPsdk adds the specified number of days to the current system time, which establishes the key's expiration date.

Syntax

```
PGPOptionListRef PGPOExpiration(
             PGPContextRef pgpContext,
             PGPUInt32 expirationDays );
```

Parameters

`pgpContext`	the target context
`expirationDays`	the desired expiration date, expressed as days from "now"

Notes

To ensure that a key or component item has no expiration date, specify `expirationDays` as having the special value `kPGPExpirationTime_Never`.

PGPOExportable

Indicate whether or not export of the key item in question is allowed. Currently, this only applies to signatures (see `PGPSignUserID`).

Syntax

```
PGPOptionListRef PGPOExportable(
             PGPContextRef pgpContext,
             PGPBoolean canExport );
```

Parameters

`pgpContext`	the target context
`canExport`	set to `TRUE` if the item is exportable

PGPOSigRegularExpression

Establishes the specified regular expression for use by `PGPSignUserID.`

Syntax

```
PGPOptionListRef PGPOSigRegularExpression(
             PGPContextRef pgpContext,
             char const *regExpr );
```

Parameters

`pgpContext`	the target context
`regExpr`	the regular expression string

Notes

This option is valid for `PGPSignUserID` only.

PGPOSigTrust

Establishes the specified signature validity for use by `PGPSignUserID`.

Syntax

```
PGPOptionListRef PGPOSigTrust(
            PGPContextRef pgpContext,
            PGPUInt32 trustLevel,
            PGPUInt32 validity );
```

Parameters

`pgpContext`	the target context
`trustLevel`	the desired trust level for signatures, which asumes `kPGPNameTrust_`... values
`validity`	the desired trust value for signatures, which asumes `kPGPValidity_`... values

Notes

This option is valid for `PGPSignUserID` only.

PGPORevocationKeySet

Option for `PGPGenerateKey()` to specify one or more Designated Revocation keys for the new key. Any of these keys will have the power to revoke the generated key without the permission or cooperation of the owner of the new key.

Syntax

```
PGPOptionListRef PGPORevocationKeySet
            PGPContextRef context,
            PGPKeySetRef raKeySetRef );
```

Parameters

`pgpContext`	the target context
`raKeySetRef`	the keys that will be able to revoke the key being generated

User Interface Dialog Option Functions

PGPOUIParentWindowHandle (*Windows platforms only*)

Indicates that the window for the associated dialog should be created as a child of the specified parent window.

Syntax

```
PGPOptionListRef PGPOUIParentWindowHandle(
            PGPContextRef pgpContext,
```

```
HWND  hwndParent );
```

Parameters

`pgpContext`	the target context
`hwndParent`	the window handle of the desired parent window

Notes

If this option is not supplied, then the dialog window is created as a child of the desktop.

PGPOUIWindowTitle

Specifies the window title text for the associated dialog.

Syntax

```
PGPOptionListRef PGPOUIWindowTitle(
             PGPContextRef pgpContext,
             const char  *title );
```

Parameters

`pgpContext`	the target context
`title`	the desired window title text

Notes

If this option is not supplied, then the window title text assumes a dialog-specific default (see the UI dialog functions in Chapter 9).

PGPOUIDialogPrompt

Specifies the prompt text for the associated dialog.

Syntax

```
PGPOptionListRef PGPOUIDialogPrompt(
             PGPContextRef pgpContext,
             const char  *prompt );
```

Parameters

`pgpContext`	the target context
`prompt`	the desired prompt text

Notes

If this option is not supplied, then the prompt text assumes a dialog-specific default (see the UI dialog functions in Chapter 9).

PGPOUIDialogOptions

Enables an options button on the associated dialog, and defines the items that will appear in the resultant options dialog window.

Currently, these items are restricted to check boxes and pop-up lists, and are specified by the `PGPOUICheckBox` and `PGPOUIPopUpList` options.

Syntax

```
PGPOptionListRef PGPOUIDialogOptions(
            PGPContextRef pgpContext,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Dialog specific options include:

- `PGPOUICheckbox`
- `PGPOUIPopupList`

Notes

The items appear in the in the resultant options dialog window in the order in which their associated option functions are specified.

PGPOUICheckBox

Describes a check box item that will appear in the resultant options dialog window of an associated `PGPOUIOptionsDialog` option function. The check box format is primarily intended to return boolean values, but provides for future return of other values. As such, an initial / resultant value of 1 (one) is considered to be `TRUE` (filled check box), while an initial / resultant value of 0 (zero) is considered to be `FALSE` (empty check box).

Syntax

```
PGPError PGPOUICheckBox(
             PGPContextRef pgpContext,
             PGPUInt32 itemID,
             const char *title,
             const char *description,
             PGPUInt32 initialValue,
             PGPUInt32 *resultValue );
```

Parameters

`pgpContext`	the target context
`itemID`	the target item
`title`	the desired title text
`description`	the desired description text (optional)
`initialValue`	the desired initilal value of the item
`resultValue`	the receiving field for the resultant value of the item

Notes

`PGPOUICheckbox()` is found in `pgpUserInterface.h`.

PGPOUIPopUpList

Describes a pop-up list item that will appear in the resultant options dialog window of an associated `PGPOUIOptionsDialog` option function. The pop-up list format allows the return of one of a list of any number of discrete values.

Initial and resultant values are indicated by their index (from zero) within the array of list values.

Syntax

```
PGPError PGPOUIPopUpList(
             PGPContextRef pgpContext,
             PGPUInt32 itemID,
             const char  *title,
             const char  *description,
             PGPUInt32 numListItems,
             const char *listItems[],
             PGPUInt32 initialValue,
             PGPUInt32 *resultValue );
```

Parameters

`pgpContext`	the target context
`itemID`	the target item
`title`	the desired title text
`description`	the desired description text (optional)
`numListItems`	the number of items in the list
`listItems`	the discrete values of the list items
`initialValue`	the index (from zero) of the desired initial list value
`resultValue`	the receiving field for the index (from zero) of the resultant list value

Notes

The items are displayed in the order in which they are specified in the list.

`PGPOUIPopUpList()` is found in `pgpUserInterface.h`.

PGPOUIOutputPassphrase

Specifies the receiving field for any resultant password collected by the employing function (usually a passphrase dialog).

Syntax

```
PGPOptionListRef PGPOUIOutputPassphrase(
             PGPContextRef pgpContext,
             char **passphrase );
```

Parameters

`pgpContext`	the target context
`passphrase`	the receiving field for the resultant passphrase

Notes

This option is required by those functions that accept it.

If the user clicks on the cancel button or the close button, then receiving field for the resultant password will reflect `NULL`.

The employing function always attempts to allocate any resultant password in secure memory (see `PGPNewSecureMemory`).

The caller is responsible for deallocating any resultant passphrase with `PGPFreeData`.

PGPOUIMinimumPassphraseLength

Establishes the minimum acceptable passphrase length (in characters) when assigning or changing a key's associated passphrase.

Syntax

```
PGPOptionListRef PGPOUIMinimumPassphraseLength(
             PGPContextRef pgpContext,
             PGPUInt32 minimumPassphraseLength );
```

Parameters

`pgpContext` the target context

`minimumPassphraseLength`

the minimum acceptable passphrase length (in bytes)

Notes

If this option is not supplied or its value is specified as zero, then any length passphrase is considered to be acceptable.

PGPOUIMinimumPassphraseQuality

Establishes the minimum acceptable passphrase quality when assigning or changing a key's associated passphrase.

Syntax

```
PGPOptionListRef PGPOUIMinimumPassphraseQuality(
             PGPContextRef pgpContext,
             PGPUInt32 minimumPassphraseQuality );
```

Parameters

`pgpContext` the target context

`minimumPassphraseQuality` the minimum acceptable estimated passphrase quality (assumes values between 0 (zero) and 100; see `PGPEstimatePassphraseQuality`)

Notes

If this option is not supplied or its value is specified as zero, then any passphrase quality is considered to be acceptable.

PGPOUIShowPassphraseQuality

Enables display of the passphrase quality "progress bar" when assigning or changing a key’s associated passphrase.

Syntax

```
PGPOptionListRef PGPOUIShowPassphraseQuality(
             PGPContextRef pgpContext,
             PGPBoolean showPassphraseQuality );
```

Parameters

`pgpContext`	the target context
`showPassphraseQuality`	set to `TRUE` to display the password quality box

Notes

If this option is not supplied, then no passphrase quality "progress bar" is displayed.

PGPOUIVerifyPassphrase

Controls passphrase verification in dialogs where a passphrase can be verified against a target key or key set. If TRUE, the passphrase dialog function will not return unless / until the user enters the correct passphrase or aborts the dialog.

Syntax

```
PGPOptionListRef PGPOUIVerifyPassphrase(
             PGPContextRef pgpContext,
             PGPBoolean verifyPassphrase );
```

Parameters

`pgpContext`	the target context
`verifyPassphrase`	set to `TRUE` to verify the passphrase

PGPOUIFindMatchingKey

Controls matching of a passphrase against keys other than the target key when `PGPOUIVerifyPassphrase` is specified.

Syntax

```
PGPOptionListRef PGPOUIFindMatchingKey(
             PGPContextRef pgpContext,
             PGPBoolean findMatchingKey );
```

Parameters

`pgpContext` the target context

`findMatchingKey`

set to TRUE to find the matching key

PGPOUIDefaultRecipients

Specifies a list of default recipients that will be initially appear in the "selected" area of a dialog that utilizies recipient lists.

Syntax

```
PGPOptionListRef PGPOUIDefaultRecipients(
             PGPContextRef pgpContext,
             PGPUInt32 numRecipients,
             const PGPRecipientSpec recipients[] );
```

Parameters

`pgpContext` the target context

`numRecipients` the number of recipients

`recipients` the array of recipients

PGPOUIRecipientGroups

Specifies a list of default recipient groups that will be initially appear in the "selected" area of a dialog that utilizies recipient lists.

Syntax

```
PGPOptionListRef PGPOUIRecipientGroups(
             PGPContextRef pgpContext,
             PGPGroupSetRef groupSet );
```

Parameters

`pgpContext` the target context

`groupSet` the group containing the desired recipients

Notes

Use multiple instances of this option to specify multiple recipient groups to a dialog. However, care in creating and maintaining groups should minimize the occasions where multiple instances are required.

PGPOUIEnforceAdditionalRecipientRequests

Specifies the desired enforcement with respect to additional recipient requests.

Syntax

```
PGPOptionListRef PGPOUIEnforceAdditionalRecipientRequests(
            PGPContextRef pgpContext,
            PGPAdditionalRecipientRequestEnforcement
               aarEnforce );
```

Parameters

`pgpContext`	the target context
`aarEnforce`	the desired enforcement policy, which assumes `kPGPARREnforcement_`... values

Notes

If this option is not supplied, then the enforcement policy assumes `kPGPARREnforcement_None`.

PGPOUIDefaultKey

Syntax

```
PGPOptionListRef PGPOUIDefaultKey(
            PGPContextRef pgpContext,
            PGPKeyRef theKey );
```

Parameters

`pgpContext`	the target context
`theKey`	the desired default encryption / signing key

PGPOUIDisplayMarginalValidity

Determines the appearance (style) of the key validity icon used whenever a dialog displays a list of keys, for example, `PGPRecipientDialog`.

A value of TRUE indicates that the dialog should use the bar-style key validity icon; a value of FALSE indicates that the dialog should use the circle-style key validity icon.

This function interacts with `PGPOUIIgnoreMarginalValidity`.

Syntax

```
PGPOptionListRef PGPOUIDisplayMarginalValidity(
            PGPContextRef pgpContext,
            PGPBoolean displayMarginalValidity );
```

Parameters

`pgpContext`	the target context
`displayMarginalValidity`	set to TRUE to display marginal validity values

PGPOUIIgnoreMarginalValidity

Determines whether or not keys that are marginally valid are displayed as such whenever a dialog displays a list of keys, for example, `PGPRecipientDialog`.

A value of as TRUE indicates that marginally valid keys should be displayed as being invalid; a value of FALSE indicates that marginally valid keys should be displayed as such.

This function interacts with `PGPOUIDisplayMarginalValidity`.

Syntax

```
PGPOptionListRef PGPOUIIgnoreMarginalValidity(
            PGPContextRef pgpContext,
            PGPBoolean ignoreMarginalValidity );
```

Parameters

`pgpContext`	the target context
`ignoreMarginalValidity`	set to TRUE to ignore marginal validity values

PGPOUIKeyServerUpdateParams

Specifies a list of key servers to search when updating missing keys in user interface dialogs.

Syntax

```
PGPOptionListRef PGPOUIKeyServerUpdateParams(
             PGPContextRef pgpContext,
             PGPUInt32 numKeyServers,
             const PGPKeyServerSpec keyServerList[],
             PGPtlsContextRef tlsContext,
             PGPBoolean searchBeforeDisplay,
             PGPKeySetRef *foundKeys,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`numKeyServers`	the number of key servers in the list
`keyServerList`	the list of key servers to search
`tlsContext`	the active TLS context
`searchBeforeDisplay`	set to TRUE if the display should appear after the search results have been obtain; set to FALSE if the display should appear while the search is in progress
`foundKeys`	the receiving field for the key set containing the resultant matching keys
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

PGPOUIKeyServerSearchFilter

Specifies a search filter to be used with key server user interface dialogs.

Syntax

```
PGPOptionListRef PGPOUIKeyServerSearchFilter(
             PGPContextRef pgpContext,
             PGPFilterRef  filter );
```

Parameters

`pgpContext`	the target context
`filter`	the desired filter

PGPOUIKeyServerSearchKey

Specifies a search key to be used with key server user interface dialogs.

Syntax

```
PGPOptionListRef PGPOUIKeyServerSearchKey(
             PGPContextRef pgpContext,
             PGPKeyRef  key );
```

Parameters

`pgpContext`	the target context
`key`	the desired key

PGPOUIKeyServerSearchKeySet

Specifies a search key set to be used with key server user interface dialogs.

Syntax

```
PGPOptionListRef PGPOUIKeyServerSearchKeySet(
             PGPContextRef pgpContext,
             PGPKeySetRef  keySet );
```

Parameters

`pgpContext`	the target context
`keySet`	the desired key set

PGPOUIKeyServerSearchKeyIDList

Specifies a search key ID list to be used with key server user interface dialogs.

Syntax

```
PGPOptionListRef PGPOUIKeyServerSearchKeyIDList(
             PGPContextRef pgpContext,
             PGPUInt32 numKeyIDs,
             const PGPKeyID keyIDList[] );
```

Parameters

`pgpContext`	the target context
`numKeyIDs`	the number of key IDs in the list
`keyIDList`	the list of keyIDs

Network and Key Server Option List Functions

PGPONetURL

Option for `PGPNewKeyServer()`, to specify the desired server by URL .

Syntax

```
PGPOptionListRef PGPONetURL(
             PGPContextRef context,
             const char *url 0;
```

Parameters

`url`	the server's URL, expressed as a null-terminated C string

PGPONetHostName

Option for `PGPNewKeyServer()`, to specify the desired server by host name and port number.

Syntax

```
PGPOptionListRef PGPONetHostName(
             PGPContextRef context,
             const char *hostName,
             PGPUInt16 port );
```

Parameters

`hostName`	the server machine's internet domain name, expressed as a null-terminated C string
`port`	the server application's port number

PGPONetHostAddress

Option for `PGPNewKeyServer()`, to specify the desired server by IP address and port number.

Syntax

```
PGPOptionListRef PGPONetHostAddress(
             PGPContextRef context,
             PGPUInt32 hostAddress,
             PGPUInt16 port );
```

Parameters

`hostAddress`	the server machine's IP address
`port`	the server application's port number

PGPOKeyServerProtocol

Option for `PGPNewKeyServer()`, to specify the protocol (i.e. HTTP, LDAP, etc.) to use when communicating with that key server.

Syntax

```
PGPOptionListRef PGPOKeyServerProtocol(
             PGPContextRef context,
             PGPKeyServerProtocol protocol );
```

Parameters

`protocol`	the desired protocol

PGPOKeyServerKeySpace

Option for `PGPNewKeyServer()`, to specify which key space (i.e. normal, pending area, or default) to examine. Note that this only applies to LDAP key servers.

Syntax

```
PGPOptionListRef PGPOKeyServerKeySpace(
             PGPContextRef context,
             PGPKeyServerKeySpace space );
```

Parameters

`space`	the desired key space

PGPOKeyServerAccessType

Option for `PGPNewKeyServer()`, to specify which kind of key server access (i.e. normal, administrator, or default) is desired. Note that this only applies to LDAP key servers.

Syntax

```
PGPOptionListRef PGPOKeyServerAccessType(
             PGPContextRef context,
             PGPKeyServerAccessType accessType );
```

Parameters

`accessType`	the desired type of access

PGPOKeyServerCAKey

Option for `PGPSendCertificateRequest()`, to address the certificate request to a particular CA key on the target host machine. Note that this option is only relevant when communicating with CA's which support more than one CA key.

Syntax

```
PGPOptionListRef PGPOKeyServerCAKey(
             PGPContextRef context,
             PGPKeyRef caKey );
```

Parameters

`caKey`	the key of the target CA

PGPOKeyServerRequestKey

Option for `PGPSendCertificateRequest()`, to supply the key for which the certificate request is being made.

Syntax

```
PGPOptionListRef PGPOKeyServerRequestKey(
             PGPContextRef context,
             PGPKeyRef requestKey );
```

Parameters

`requestKey`	the key for which you're requesting the certificate

PGPOKeyServerSearchKey

Option for `PGPRetrieveCertificateRequest()`, to specify the key to retrieve (i.e., the key for which an earlier certificate request was made).

Syntax

```
PGPOptionListRef PGPOKeyServerSearchKey(
             PGPContextRef context,
             PGPKeyRef searchKey );
```

Parameters

`searchKey`	the key to retrieve

PGPOKeyServerSearchFilter

Option for `PGPRetrieveCertificateRequest()`, to specify how to search for the key(s) to retrieve (i.e., keys for which earlier certificate requests were made). A filter can search for keys based on key properties, for example a particular key ID.

Syntax

```
PGPOptionListRef PGPOKeyServerSearchFilter(
             PGPContextRef context,
             PGPFilterRef searchFilter );
```

Parameters

`searchFilter`	the filter to use when searching

Misc. Option List Functions

PGPONullOption

Returns a special `PGPOptionListRef` that is always ignored.

Syntax

```
PGPOptionListRef PGPONullOption(
             PGPContextRef pgpContext );
```

Notes

While this function is useful for providing a placeholder or default value in dynamically constructed option lists, the same results can be achieved by assembling the dynamic option list from modular, persistent lists.

PGPOCompression

Indicates whether or not the input plain text should be compressed prior to encrypting or signing in binary format.

Syntax

```
PGPOptionListRef PGPOCompression(
             PGPContextRef pgpContext,
             PGPBoolean isCompressed );
```

Parameters

`pgpContext`	the target context
`isCompressed`	set to `TRUE` to indicate compress plain text before encrypting or signing

Notes

This option should routinely be specified as `TRUE`, since prior compression will not only reduce the size of the resultant cipher text, but also will increase the strength of the cipher text in most cases. This increase in the strength is partially a result of the reduction in plain text character frequency, and partially a result of the reduction in the amount of resultant cipher text.

Strong cipher text is essentially immune to compression, since it has large numbers of distinct “characters” that rarely if ever form repeating sequences.

PGPOCommentString

Indicates that the specified comment string should be included in the message blocks.

Syntax

```
PGPOptionListRef PGPOCommentString(
            PGPContextRef pgpContext,
            char const *commentString );
```

Parameters

`pgpContext`	the target context
`commentString`	the comment text

PGPOVersionString

Indicates that the specified version string should be included in the message blocks.

Syntax

```
PGPOptionListRef PGPOVersionString(
            PGPContextRef pgpContext,
            char const *versionString );
```

Parameters

`pgpContext`	the target context
`versionString`	the desired version string

PGPOPassphrase

Specifies the passphrase to be used for signing, conventional encrypting, and decrypting.

Syntax

```
PGPOptionListRef PGPOPassphrase(
            PGPContextRef pgpContext,
            const char *passphraseBuf );
```

Parameters

`pgpContext`	the target context
`passphraseBuf`	the passphrase string

Notes

For signing and conventional encryption, this option must be specified as a sub-option (see `PGPOSignWIthKey` and `PGPOConventionalEncrypt`).

PGPOPassphraseBuffer

Specifies the passphrase to be used for signing, conventional encrypting, and decrypting. This differs from `PGPOPassphrase` in that the passphrase data and length are arbitrary, rather than being constrained to a *C* language string.

Syntax

```
PGPOptionListRef PGPOPassphraseBuffer(
             PGPContextRef pgpContext,
             const void *passphraseBuf,
             PGPSize passphraseLength );
```

Parameters

`pgpContext`	the target context
`passphraseBuf`	the passphrase data
`passphraseLength`	the length of the passphrase data (in bytes)

Notes

For signing and conventional encryption, this option must be set as a sub-option (see `PGPOSignWIthKey` and `PGPOConventionalEncrypt`).

PGPOPasskeyBuffer

Specifies the passkey to be used for signing, conventional encrypting, and decrypting. This function is similar to `PGPOPassphrase` and `PGPOPassphraseBuffer`, but for keys having shares (that is, “split” keys). The actual passkey data and length are those returned from a key reconstitution dialog.

Syntax

```
PGPOptionListRef PGPOPasskeyBuffer(
             PGPContextRef pgpContext,
             const void *passkeyBuf,
             PGPSize passkeyLength );
```

Parameters

`pgpContext`	the target context
`passkeyBuf`	the passkey data
`passkeyLength`	the length of the passkey data (in bytes)

Notes

For signing and conventional encryption, this option must be set as a sub-option (see `PGPOSignWIthKey` and `PGPOConventionalEncrypt`).

PGPOPreferredAlgorithms

Establishes the specified symmetric cipher algorithm(s) as the preferred algorithm(s) to use when generating keys and their sub-items, as well as when encrypting and signing. The order of the array determines the relative preferences, with the first element in the array being the most preferred algorithm.

Syntax

```
PGPOptionListRef PGPOPreferredAlgorithms(
            PGPContextRef pgpContext,
            PGPCipherAlgorithm const *cipherKeyAlg,
            PGPUInt32 cipherKeyAlgCount );
```

Parameters

`pgpContext`	the target context
`cipherKeyAlg`	an array of the preferred symmetric cipher algorithms
`cipherKeyAlgCount`	the number of symmetric cipher algorithms in the ordered array

Notes

The number of symmetric cipher algorithms in the ordered array must be between one and the number of available symmetric cipher algorithms (see `PGPCountSymmetricCiphers`).

No assumption is made regarding the actual availability of the symmetric cipher algorithm(s) listed in the array.

The actual choice of algorithm involves availability and acceptability considerations; this function simply adds a preference consideration.

PGPOKeySetRef

For signature **validation** and decryption operations, use the *key database associated with* the specified key set as the look-up source for signature and decryption keys.

For key generation operations, use the *key database associated with* the specified key set as the destination for newly generated keys.

This option is required by those functions accepting it.

Syntax

```
PGPOptionListRef PGPOKeySetRef(
             PGPContextRef pgpContext,
             PGPKeySetRef keySet );
```

Parameters

`pgpContext`	the target context
`keySet`	the desired key set

Notes

The current implementation treats the specified key set as an indirect parameter that references a key database, rather than as an explicit destination.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to the function's semantics and usage.

PGPOSendNullEvents

Post a null event at each specified interval. This interval is approximate, but is guaranteed never to be less than that specified.

Syntax

```
PGPOptionListRef PGPOSendNullEvents(
             PGPContextRef pgpContext,
             PGPTimeInterval approxInterval );
```

Parameters

`pgpContext`	the target context
`approxInterval`	the desired time interval (in milliseconds) between event postings

Notes

These events provide a mechanism and a data source for implementing progress bars, as well as a window of opportunity to pause, modify, or terminate the job.

PGPOX509Encoding

Indicates whether or not the associated input buffer / file should be considered to be an ANSI X.509 certificate, rather than a key set in PGP export format. Currently, specifying this option with the `x509Encoding` argument set to `TRUE` results in a singleton key set containing an *unsigned* key whose name is based on the `commonName` portion of the distinguished name in the certificate.

Syntax

```
PGPOptionListRef PGPOX509Encoding(
             PGPContextRef pgpContext,
```

```
                    PGPBoolean x509Encoding );
```

Parameters

`pgpContext`	the target context
`x509Encoding`	set to `TRUE` if the associated input buffer / file should be considered as an ANSI X.509 certificate

Notes

This option is valid for `PGPImportKeySet` only.

If this option is *not* specified, then `PGPImportKeySet` treats its associated input buffer / file as a key set in PGP export format, which maintains compatibility with previous PGPsdk versions.

Future PGPsdk versions may modify this option to yield a valid *signed* PGP key based upon information in the certificate.

PGPOExportFormat

Option for `PGPExport()` to specify the desired export data format. `PGPExport()` can export either keys or additional items, such as certificate and CRL request messages. For a list of all available `PGPExportFormat` values, please see `pgpOptionList.h`.

Syntax

```
PGPOptionListRef PGPOExportFormat(
                    PGPContextRef pgpContext,
                    PGPExportFormat exportFormat );
```

Parameters

`pgpContext`	the target context
`exportFormat`	the desired export format

PGPOExportPrivateSubkeys

Option for `PGPExport()` to control whether or not private subkeys are included in the exported data.

Syntax

```
PGPOptionListRef PGPOExportPrivateSubkeys(
                    PGPContextRef context,
                    PGPBoolean exportSubkeys );
```

Parameters

`pgpContext`	the target context
`exportSubkeys`	set to `TRUE` to enable output of private subkeys in the

exported data

PGPOEventHandler

Establish the specified function as the user event handler.

Syntax

```
PGPOptionListRef PGPOEventHandler(
            PGPContextRef pgpContext,
            PGPEventHandlerProcPtr eventHandler,
            PGPUserValue eventHandlerArg );
```

Parameters

`pgpContext`	the target context
`eventHandler`	the desired event handler
`eventHandlerArg`	the user-defined data to be passed as an argument to the event handler

Notes

For greatest flexibility, the PGPsdk developer should consider establishing `eventHandlerArg` as a pointer to a user-defined data type, for example a *C* `struct`.

Specify `eventHandlerArg` as `( PGPUserData )0` to indicate a dummy argument.

PGPOLastOption

All functions having a variable number of arguments must include a special argument to indicate the end of the argument list. This function provides that argument, and *must* appear at the end of every variable argument list.

Syntax

```
PGPOptionListRef PGPOLastOption(PGPContextRef pgpContext);
```

Parameters

`pgpContext`	the target context

Group Functions 4

Introduction

The group management functions provide utilities for manipulating named lists of key IDs. Groups can contain other groups. Functions are provided for resolving groups into key sets for use with encoding functions. At this time, groups are a higher level concept not directly supported by most of the PGPsdk APIs.

Group Set Management Functions

PGPNewGroupSet

Creates a new, empty collection of groups.

Syntax

```
PGPError PGPNewGroupSet(
             PGPContextRef pgpContext,
             PGPGroupSetRef *groupSet );
```

Parameters

`pgpContext`	the target context
`groupSet`	the receiving field for the resultant group set

Notes

The caller is responsible for de-allocating the resultant group set with `PGPFreeGroupSet`.

PGPNewGroupSetFromFile (*Non-MacOS platforms only*)

Creates a new collection of groups from the specified file data.

Syntax

```
PGPError PGPNewGroupSetFromFile(
             PGPContextRef pgpContext,
             PGPFileSpecRef fileSpec,
             PGPGroupSetRef *groupSet );
```

Parameters

`pgpContext`	the target context
`fileSpec`	the source file specification
`groupSet`	the receiving field for the resultant group set

Notes

`fileSpec` is assumed to reference a file that was created by `PGPSaveGroupSetToFile`.

The caller is responsible for de-allocating the resultant group set with `PGPFreeGroupSet`.

PGPNewGroupSetFromFSSpec (*MacOS platforms only*)

Creates a new collection of groups from the specified file data.

Syntax

```
PGPError PGPNewGroupSet(
             PGPContextRef pgpContext,
             const FSSpec *spec,
             PGPGroupSetRef *groupSet );
```

Parameters

`pgpContext`	the target context
`spec`	the source Macintosh FS specification
`groupSet`	the receiving field for the resultant group set

Notes

`spec` is assumed to reference a file that was created by `PGPSaveGroupSetToFile`.

The caller is responsible for de-allocating the resultant group set with `PGPFreeGroupSet`.

PGPCopyGroupSet

Creates an exact copy of the source group set.

Syntax

```
PGPError PGPCopyGroupSet(
             PGPGroupSetRef srcSet,
             PGPGroupSetRef *destSet );
```

Parameters

`srcSet`	the source group set
`destSet`	the receiving field for the copy of the group set

Notes

The caller is responsible for de-allocating the resultant group set copy with `PGPFreeGroupSet`.

PGPFreeGroupSet

Frees the specified collection of groups.

Syntax

```
PGPError PGPFreeGroupSet( PGPGroupSetRef groupSet );
```

Parameters

`groupSet`	the target group set

Notes

Group sets do *not* have associated reference counts – the data item is always de-allocated.

PGPGetGroupSetContext

Returns the context associated with the specified collection of groups.

Syntax

```
PGPContextRef PGPGetGroupSetContext(
             PGPGroupSetRef groupSet );
```

Parameters

`groupSet`	the target group set

Notes

If the specified group set is not valid, then the returned context reference value is set to `kInvalidPGPContextRef`.

PGPGroupSetNeedsCommit

Returns TRUE if the contents of the in-memory collection of groups has changed in any way, and so should be written to disk to make those changes permanent (see `PGPSaveGroupSetToFile`).

Syntax

```
PGPBoolean PGPGroupSetNeedsCommit(
             PGPGroupSetRef groupSet );
```

Parameters

`groupSet`	the target group set

PGPSaveGroupSetToFile

Saves the in-memory collection of groups to the specified file.

Syntax

```
PGPError PGPSaveGroupSetToFile(
             PGPGroupSetRef groupSet,
             PGPFileSpecRef fileSpec );
```

Parameters

`groupSet`	the source group set
`fileSpec`	the specification of the desired output file

Notes

Any existing file is silently overwritten.

This function should only be called if `PGPGroupSetNeedsCommit` returns `TRUE`.

PGPExportGroupSetToBuffer

Transfers an in-memory collection of groups to a dynamically allocated buffer.

Syntax

```
PGPError PGPExportGroupSetToBuffer(
             PGPGroupSetRef groupSet,
             void **groupData,
             PGPSize *groupDataLength );
```

Parameters

`groupSet`	the source group set
`groupData`	the receiving field for the pointer to the allocated group data buffer
`groupDataLength`	the receiving field for the resultant length of the group data (in bytes)

Notes

The caller is responsible for de-allocating the resultant group data buffer with `PGPFreeData`.

PGPImportGroupSetFromBuffer

Populates an in-memory collection of groups from the data in the specified buffer.

Syntax

```
PGPError PGPImportGroupSetFromBuffer(
             PGPContextRef pgpContext,
             void *groupData,
             PGPSize groupDataLength,
             PGPGroupSetRef *groupSet );
```

Parameters

`pgpContext`	the target context
`groupData`	the buffer containing the group data
`groupDataLength`	the length of the group data (in bytes)
`groupSet`	the receiving field for the resultant group set

Notes

The data in the specified is expected to be in the format created by `PGPExportGroupSetToBuffer`.

The caller is responsible for de-allocating the resultant group set with `PGPFreeGroupSet`.

PGPMergeGroupSets

Merge the specified source group set into the specified destination group set.

Syntax

```
PGPError PGPMergeGroupSets(
             PGPGroupSetRef srcSet,
             PGPGroupSetRef destSet );
```

Parameters

`srcSet`	the source group set
`destSet`	the destination group set

PGPSortGroupSetStd

Perform a standard name sort on the specified group.

Syntax

```
PGPError PGPSortGroupSetStd(
             PGPGroupSetRef groupSet,
             PGPKeySetRef keys );
```

Parameters

`groupSet`	the target group set
`keys`	the target key set

PGPSortGroupSet

Sort the items (groups and key ID's) in the target group set according to the specified comparison function.

Syntax

```
PGPError PGPSortGroupSet(
             PGPGroupSetRef groupSet,
             PGPGroupItemCompareProc compareProc,
             PGPUserValue userValue );
```

Parameters

`groupSet`	the target group set
`compareProc`	sort comparison function
`userValue`	user-defined data

PGPCountGroupsInSet

Returns the number of groups currently in the specified group set.

Syntax

```
PGPError PGPCountGroupsInSet(
             PGPGroupSetRef groupSet,
             PGPUInt32 *numGroups );
```

Parameters

`groupSet`	the target group set
`numGroups`	the resultant count

PGPGetIndGroupID

Retrieve the group ID of the n^{th} group in the specified group set.

Syntax

```
PGPError PGPGetIndGroupID(
             PGPGroupSetRef groupSet,
             PGPUInt32 groupIndex,
             PGPGroupID *groupID );
```

Parameters

`groupSet`	the target group set
`groupIndex`	the index (from zero) of the target group in the set
`groupID`	the receiving field for the resultant group ID

Group Management Functions

PGPNewGroup

Creates a new, empty group, and associates it with the specified group set.

Syntax

```
PGPError PGPNewGroup(
             PGPGroupSetRef groupSet,
             const char *name,
             const char *description,
             PGPGroupID *groupID );
```

Parameters

`groupSet`	the target group set
`name`	the value for the `name` member of the resultant group's `PGPGroupInfo` data
`description`	the value for the description member of the resultant group's `PGPGroupInfo` data
`groupID`	the receiving field for the resultant group ID

Notes

The length of the `name` argument must *not* exceed `kPGPMaxGroupNameLength`.

The length of the `description` argument must *not* exceed `kPGPMaxGroupdescriptionLength`.

The group is automatically de-allocated when its associated group set is freed with `PGPFreeGroupSet`.

PGPDeleteGroup

Removes the specified group from the specified group set.

Syntax

```
PGPError PGPDeleteGroup(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID );
```

Parameters

`groupSet`	the target group set
`groupID`	the group ID of the target group in the set

Notes

The resultant group is de-allocated when its associated group set is freed with `PGPFreeGroup`.

PGPAddItemToGroup

Add the specified item to the specified group. This may be either another group (`kPGPGroupItem_Group`) or a key (`kPGPGroupItem_KeyID`).

Syntax

```
PGPError PGPAddItemToGroup(
             PGPGroupSetRef groupSet,
             PGPGroupItem const *item,
             PGPGroupID group );
```

Parameters

`groupSet`	the target group set
`description`	the target item to add
`groupID`	the target group in the set

Notes

All fields of the specified `PGPGroupItem` *must* be set.

PGPSetGroupName

Set the `name` of the target group to that specified.

Syntax

```
PGPError PGPSetGroupName(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             const char *name );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group in the set
`name`	the desired name string

Notes

The length of the `name` argument must *not* exceed `kPGPMaxGroupNameLength`.

PGPSetGroupDescription

Set the `description` of the target group to that specified.

Syntax

```
PGPError PGPSetGroupDescription(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             const char *description );
```

Parameters

`groupSet`	the target group set
`groupID`	the group ID of the target group in the set in the set
`description`	the desired description string

Notes

The length of the `description` argument must *not* exceed `kPGPMaxGroupDescriptionLength`.

PGPSetGroupUserValue

Set the user-defined data of the target group to that specified.

Syntax

```
PGPError PGPSetGroupUserValue(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPUserValue userValue );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group in the set
`userValue`	the desired user-defined data

PGPGetGroupInfo

Retrieve the information for the specified group.

Syntax

```
PGPError PGPGetGroupInfo(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPGroupInfo *info );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group in the set
`info`	the receiving field for the resultant group information

PGPSortGroupItems

Sort the item in the specified group according to the specified comparison function.

Syntax

```
PGPError PGPSortGroupItems(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPGroupItemCompareProc compareProc,
             PGPUserValue userValue );
```

Parameters

`groupSet`	the target group set
`groupID`	the group ID of the target group in the set
`compareProc`	sort comparison function
`userValue`	the desired user-defined data

PGPCountGroupItems

Determines the number of items in the specified groups.

Syntax

```
PGPError PGPCountGroupItems(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPBoolean recursive,
             PGPUInt32 *numKeys,
             PGPUInt32 *totalItems );
```

Parameters

`groupSet`	the target group set
`groupID`	the group ID of the target group in the set
`recursive`	indicates whether or not to expand any items that are groups
`numKeys`	the resultant count of key items
`totalItems`	the resultant count of all items (keys and groups)

PGPSetIndGroupItemUserValue

Sets the user-defined data of the n^{th} item in the target group to that specified. The item may be a key or a sub-group.

Syntax

```
PGPError PGPSetIndGroupItemUserValue(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPUInt32 groupIndex,
             PGPUserValue userValue );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group ID
`groupIndex`	the index (from zero) of the target item
`userValue`	the desired user-defined data

PGPGetIndGroupItem

Retrieve the n^{th} item in the specified group, which may be a key or a sub-group.

Syntax

```
PGPError PGPGetIndGroupItem(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPUInt32 groupIndex,
             PGPGroupItem *itemRef );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group ID
`groupIndex`	the index (from zero) of the target item
`itemRef`	the receiving field for the resultant item

PGPDeleteItemFromGroup

Delete the target item from the specified group.

Syntax

```
PGPError PGPDeleteItemFromGroup(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPGroupItem const *itemRef );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group ID
`itemRef`	the target item

PGPDeleteIndItemFromGroup

Delete the n^{th} item in the specified group.

Syntax

```
PGPError PGPDeleteIndItemFromGroup(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPUInt32 groupIndex );
```

Parameters

`groupSet`	the target group set
`groupID`	the target group ID
`groupIndex`	the index (from zero) of the target item

PGPMergeGroupIntoDifferentSet

Merge the specified group into the specified destination group set.

Syntax

```
PGPError PGPMergeGroupIntoDifferentSet(
             PGPGroupSetRef srcSet,
             PGPGroupID srcGroupID,
```

```
PGPGroupSetRef destSet );
```

Parameters

`srcSet`	the source group set
`srcGroupID`	the group ID of the target group in the set
`destSet`	the destination group set

Group Item Iteration Functions

PGPNewGroupItemIter

Creates a new iterator on a group for the specified item type(s). Unlike the key iterators (see the `PGPKeyIter`... functions), this is *not* a full-fledged iterator: you may not add or delete items while iterating, and you may only move forward. However, you may change the values of items.

Syntax

```
PGPError PGPNewGroupItemIter(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPGroupItemIterFlags flags,
             PGPGroupItemIterRef *iter );
```

Parameters

`groupSet`	the target group set
`groupID`	the group ID of the target group in the set
`flags`	the item specifier, which assumes `kPGPGroupIterFlags...` values
`iter`	the receiving field for the iterator

Notes

The caller is responsible for de-allocating the resultant iterator with `PGPFreeGroupItemIter`.

PGPFreeGroupItemIter

De-allocates the specified group item iterator.

Syntax

```
PGPError PGPFreeGroupItemIter( PGPGroupItemIterRef iter );
```

Parameters

`iter`	the target iterator

PGPGroupItemIterNext

Advances the specified iterator and places the data associated with the next group item into the specified receiving field.

Syntax

```
PGPError PGPGroupItemIterNext(
             PGPGroupItemIterRef iter,
             PGPGroupItem   *item );
```

Parameters

`iter`	the target iterator
`item`	the receiving field for the resultant item

Notes

Returns `kPGPError_EndOfIteration` when at the end of the group's items.

Group Utility Functions

PGPGetGroupLowestValidity

Returns the lowest validity of any item in the group. `keySet` should contain all keys available. It is not an error if keys can not be found; you may want to check the not found count.

Syntax

```
PGPError PGPGetGroupLowestValidity(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPKeySetRef keySet,
             PGPValidity *lowestValidity,
             PGPUInt32 *numKeysNotFound );
```

Parameters

`groupSet`	the target source group set
`groupID`	the group ID of the target group in the set
`keySet`	the reference key set
`lowestValidity`	the receiving field for the resultant lowest validity
`numKeysNotFound`	the receiving field for the number of keys not found

Notes

The lowest validity is `kPGPValidity_Invalid`; `kPGPValidity_Unknown` is never returned.

The current implementation treats the supplied key set as an indirect parameter that references a key database, rather than as an explicit source key set.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPNewKeySetFromGroup

Creates a new key set on the *key database associated with* the specified key set, and populates it with the keys contained in the specified group and its sub-groups.

Syntax

```
PGPError PGPNewKeySetFromGroup(
             PGPGroupSetRef groupSet,
             PGPGroupID groupID,
             PGPKeySetRef keySet,
             PGPKeySetRef *resultSet,
             PGPUInt32 *numKeysNotFound );
```

Parameters

`groupSet`	the target group set
`groupID`	the group ID of the target group in the set
`keySet`	the destination group set
`resultSet`	the receiving field for the resultant key set
`numKeysNotFound`	the receiving field for the number of keys not found

Notes

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

The current implementation treats the supplied key set as an indirect parameter that references a key database, rather than as an explicit source key set.

The indirect nature of this interface is likely to change in a future version, and will almost certainly involve changes to this function's parameterization.

PGPNewFlattenedGroupFromGroup

Create a new, simple, flattened group of unique key ID's from the specified source group, places it into the specified destination group set, and assigns it a group ID.

Syntax

```
PGPError PGPNewFlattenedGroupFromGroup(
             PGPGroupSetRef srcSet,
             PGPGroupID srcGroupID,
             PGPGroupSetRef destSet,
             PGPGroupID *destID );
```

Parameters

`srcSet`	the target source group set
`srcGroupID`	the group ID of the target group in the set
`destSet`	the destination group set
`destID`	the receiving field for the group ID of the resultant flattened group

Notes

The caller is responsible for de-allocating the resultant group with `PGPDeleteGroup`.

`srcSet` and `destSet` may *not* refer to the same group set.

Ciphering and Authentication Functions 5

Introduction

The PGPsdk provides high-level, algorithm-independent cryptographic functions for encrypting, decrypting, hashing, signing, and verifying messages and data. These not only free applications from having to be aware of the particular algorithm being used, but also allow new algorithms to be supported as they become available. Function prototypes are listed in the public header file `pgpEncode.h`. In most cases, inputs and outputs can be specified as any arbitrary combination of memory buffers and / or data files.

The PGPsdk also provides low-level cryptographic functions for developers who have special requirements, or require greater control over ciphering and authentication activities, since the high-level functions are based on **cipher feedback mode** methodology.

Certain PGPsdk functions – most notably decryption and key generation (see Chapter 2, “Key Management Functions.”)—require a significant amount of time to complete. To facilitate control and progress tracking, these functions support an event and callback mechanism. This same mechanism also provides for prompting of required information when required for example, file specifications, passphrases.

Header Files

`pgpCBC.h`

`pgpCFB.h`

`pgpEncode.h`

`pgpHash.h`

`pgpHMAC.h`

`pgpPublicKey.h`

`pgpSymmetricCipher.h`

Events and Callbacks

The PGPOEventHandler option allows the calling application to request callbacks when various events occur, and to define the function (event handler) that is the target of the callback. While an event handler is usually not needed for encryption operations, it is often needed for decryption operations.

An event handler serves two purposes – it provides notification to the calling application that an event has occurred, and provides a mechanism for the calling application to affect processing (in a pre-defined manner). Notification includes a pointer to a PGPEvent data type that, depending on the type of event, provides detailed information about the cause of the event. The calling application can then respond appropriately, which may or may not intervene and affect the course of further processing. If the calling application wishes to intervene, then it can abort the job by returning an error code (a value other than kPGPError_NoErr, except in the cases of kPGPEvent_ErrorEvent). Additionally, depending on the type of event, it can modify the processing context by invoking PGPAddJobOptions.

All event handlers are declared as

```
PGPError myEvents( PGPContextRef pgpContext,
    PGPEvent *event,
    PGPUserValue userValue );
```

The pgpContext argument is the reference to the context of the job posting the event. The event argument references a PGPEvent data type as follows:

```
struct PGPEvent_
{
    PGPVersion version;
    struct PGPEvent_ *nextEvent;
    PGPJobRef job;
    PGPEventType type;
    PGPEventData data;
};
typedef struct PGPEvent_ PGPEvent;
```

The version and nextEvent members are currently reserved for internal use. The job member references the currently active encode or decode activity. The type member identifies the event being posted. The data member is a union of the event-specific data structures, which are described with their corresponding event (some events have no associated event-specific data).

The calling application can modify the processing context by invoking PGPAddJobOptions as:

```
PGPError PGPAddJobOptions( PGPJobRef job, ... );
```

The value of the job argument is that of the PGPEvent argument's job member. Additional PGPOptionListRef arguments are specified similarly to the way they are passed to PGPEncode and PGPDecode. However, only certain options can be set after each type of event, and these are listed for each event.

Figure 5-1. Encode Processing Event Sequence

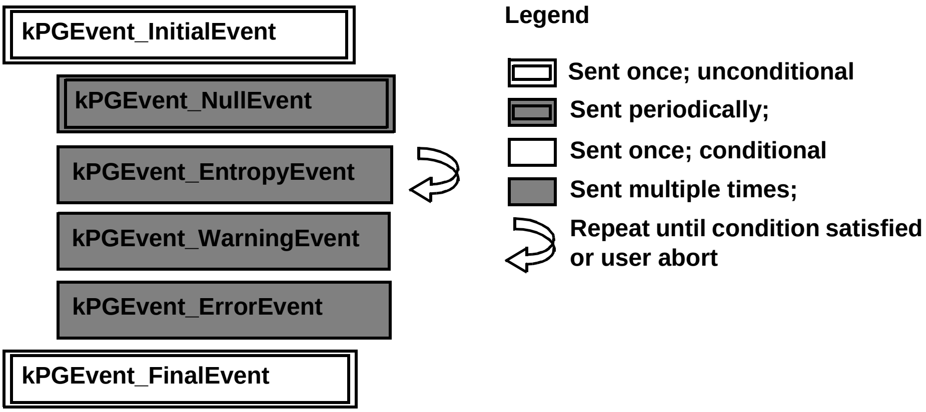
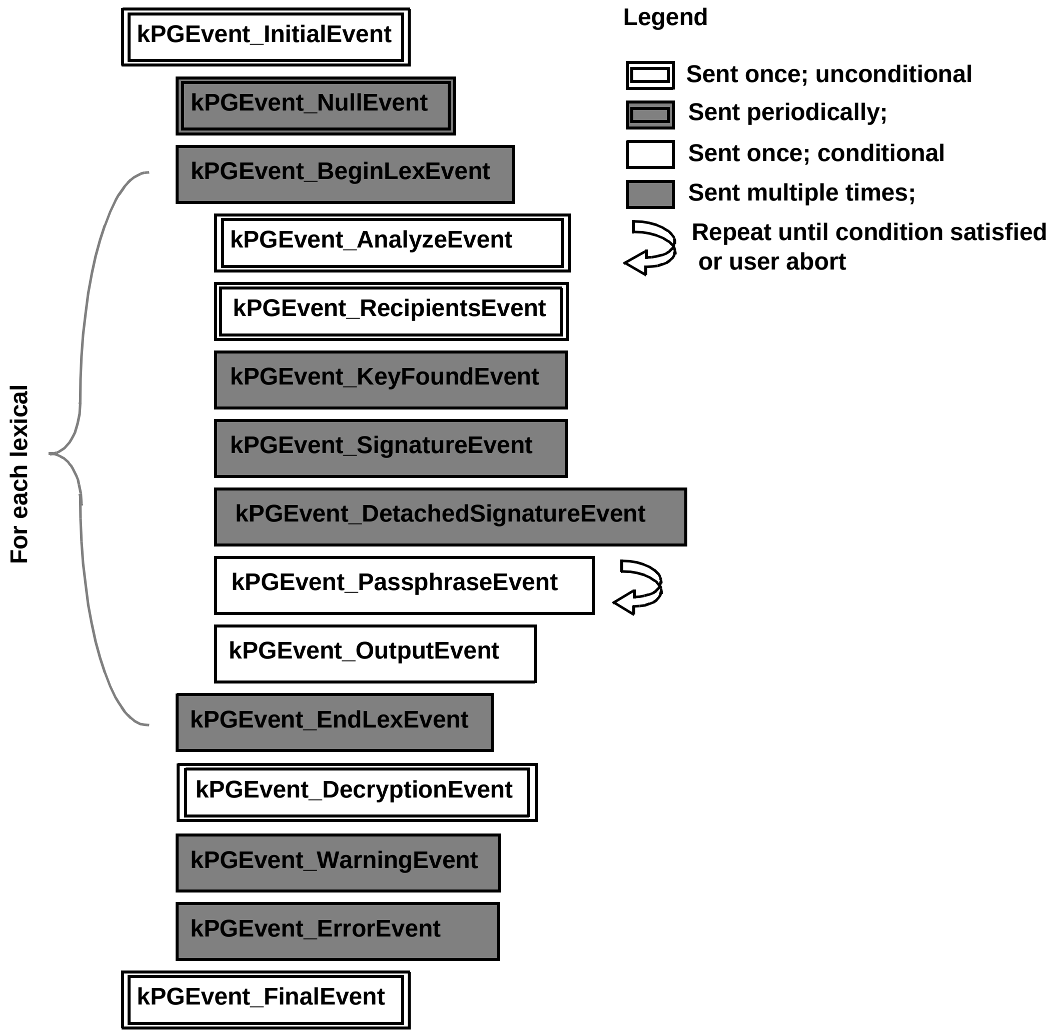


Figure 5-2. Decode Processing Event Sequence



Common Cipher Events

kPGPEvent_InitialEvent

Sent before all other events. Implies initiation of the job.

Data

None

kPGPEvent_NullEvent

Sent during the course of encode / decode processing if explicitly requested with `PGPOSendNullEvents` (see `PGPEncode` and `PGPDecode`).

The event data allows the PGPsdk developer to determine the sending function's progress and completion percentage. Its members should be treated as relative, un-scaled quantities – they are not necessarily byte quantities.

Progress tracking that involves compressed input files is rarely linear, since it tracks access of the compressed data, and not the decompression and processing of the resultant expanded data.

Data

```
typedef struct PGPEventNullData_
{
    PGPFileOffset bytesWritten;
    PGPFileOffset bytesTotal;
} PGPEventNullData;
```

kPGPEvent_WarningEvent

Sent whenever a non-fatal error occurs during processing. The associated event data always includes the error code, and for certain warnings includes an error-specific argument.

Data

```
typedef struct PGPEventWarningData_
{
    PGPError warning;
    void *warningArg;
} PGPEventWarningData;
```

kPGPEvent_ErrorEvent

Sent whenever a fatal error occurs during processing. The associated event data always includes the error code, and for certain errors includes an error-specific argument. Upon return from the event handler, the job will always abort and return the initial error code – the value returned by the event handler is ignored.

Data

```
typedef struct PGPEventErrorData_
{
    PGPError error;
    void *errorArg;
} PGPEventErrorData;
```

kPGPEvent_FinalEvent

Sent after all other events. Implies termination of the job.

PGPEncode-only Events

kPGPEvent_EntropyEvent

Sent if more entropy is needed for signing or encrypting, and indicates the minimum number of entropy bits that the event handler should add to the random pool (see Chapter 8, "Global Random Number Pool Management Functions.", for descriptions of the available random number pool management functions). For example:

```
while ( !PGPGlobalRandomPoolHasMinimumEntropy( void ) )
{
    PGPGlobalRandomPoolAddKeystroke(
    myGetKeystrokeFunction( void ) );
}
```

Data

```
typedef struct PGPEventEntropyData_
{
    PGPUInt32 entropyBitsNeeded;
} PGPEventEntropyData;
```

PGPDecode-only Events

kPGPEvent_BeginLexEvent

Sent whenever a new **lexical section** is encountered in the input. A PGP lexical section is a block of data delimited by `---BEGIN PGP` and `---END PGP` (ASCII input; binary input has only one section). A lexical section can also be a block of data before, between, or after `---BEGIN PGP` and `---END PGP` which contains no PGP data. The zero-based `sectionNumber` value indicates which section has been encountered.

Data

```
typedef struct PGPEventBeginLexData_
{
    PGPUInt32 sectionNumber;
    PGPSize sectionOffset;
} PGPEventBeginLexData;
```

kPGPEvent_AnalyzeEvent

Sent immediately after a `BeginLexEvent` to identify the type of the current lexical section. This allows the event handler to decide if it should skip this lexical section, but not abort the whole job, by returning the special error value `kPGPError_SkipSection`.

Data

```
typedef struct PGPEventAnalyzeData_
{
    PGPAnalyzeType sectionType;
} PGPEventAnalyzeData;
```

kPGPEvent_RecipientsEvent

Sent immediately after an `AnalyzeEvent` to describe the recipient(s) of the message. Generally, there can be three types of recipients:

- keys that are on the active key ring
- keys that are *not* on the active key ring
- conventional encryption passphrases

Determination of which keys are present is based upon a search of the key set specified in the `PGPOKeySetRef` option passed to `PGPDecode`. Generally, this key set will have resulted from opening the default key ring (see `PGPOpenDefaultKeyRings`, `PGPOpenKeyRing`, and `PGPOpenKeyRingPair`).

`recipientSet` identifies the set of keys required to decrypt the message, and which are currently available. `conventionalPassphraseCount` indicates how many different passphrases the message is encrypted to (typically zero or one). `keyCount` indicates the number of keys required to decrypt the message that are not currently available, and these are identified by `keyID` in the referenced `keyIDArray`.

Data

```
typedef struct PGPEventRecipientsData_
{
    PGPKeySetRef recipientSet;
    PGPUInt32 conventionalPassphraseCount;
    PGPUInt32 keyCount;
    PGPKeyID const *keyIDArray;
} PGPEventRecipientsData;
```

kPGPEvent_KeyFoundEvent

Sent whenever all of the following are TRUE:

- a key is found in the input data
- the PGPOImportKeysTo option was *not* specified, telling the job where to put the key
- the PGPOSendEventIfKeyFound option was specified

keySet holds the key found in the input data, and this key set is automatically freed upon return. The event handler code can process the key in anyway it sees fit, but will usually choose to merge the key into some key set (see PGPAddKeys).

Data

```
typedef struct PGPEventKeyFoundData_
{
    PGPKeySetRef keySet;
} PGPEventKeyFoundData;
```

kPGPEvent_SignatureEvent

Sent for signed messages to provide information about the signature status.

signingKeyID always contains the key ID of the signing key. signingKey contains the signing key itself if it is in the key set passed to PGPDecode.

The key validity flags increase monotonically, that is, if one is TRUE, then the flags preceding it must also be TRUE:

- checked indicates that the key is available, and that the message is properly formatted
- verified indicates that the signature validated correctly
- keyRevoked, keyDisabled, and keyExpired indicate that the signing key is no longer active
- keyValidity indicates the validity level of the signing key

The keyValidity flag is set based on the signing key's validity in relation to the thresholds set by the PGPDecode options PGPOWarnBelowValidity and PGPOFailBelowValidity.

creationTime indicates when the key was signed.

Data

```
typedef struct PGPEventSignatureData_
{
    PGPKeyID signingKeyID;
    PGPKeyRef signingKey;
    PGPBoolean checked;
    PGPBoolean verified;
    PGPBoolean keyRevoked;
    PGPBoolean keyDisabled;
    PGPBoolean keyExpired;
    PGPBoolean keyMeetsValidityThreshold;
    PGPValidity keyValidity;
    PGPTime creationTime;
} PGPEventSignatureData;
```

kPGPEvent_DetachedSigEvent

Sent to notify the event handler that the input file contains a detached signature (a signature that is not attached to the file it signs). The event handler must provide an input source to be signature-checked against the detached signature. This can be any of the forms of input described among the options. The event handler should invoke `PGPAddJobOptions` specifying the `PGPODetachedSig` option with the input data to be checked as a sub-option.

Data

None

kPGPEvent_PassphraseEvent

Sent if a passphrase is needed for decrypting (posted by `PGPDecode`), either to unlock a decryption key or to decrypt a conventionally encrypted message. The event handler should obtain an appropriate passphrase, perhaps by interacting with the user to get a typed-in passphrase, and then invoke `PGPAddJobOptions` specifying the `PGPOPassphrase`, `PGPOPassphraseBuffer`, or `PGPOPasskeyBuffer` option, or return `kPGPError_UserAbort` if no passphrase is available.

If a passphrase is needed for a conventionally encrypted message, then the `fConventional` flag is `TRUE`, and `keyset` is ignored. Otherwise, `keyset` includes the key(s) for which a passphrase is needed.

If a passphrase is needed for decryption, then `keyset` will hold multiple keys if multiple secret keys on the key ring can decrypt the message . However, any passphrase that unlocks any of these secret keys is acceptable as a response.

This event is sent repeatedly until a valid passphrase is received, or until the event handler requests abort of the job. This allows the event handler to enforce a limit on the number of passphrase attempts.

Data

```
typedef struct PGPEventPassphraseData_
{
    PGPBoolean fConventional;
    PGPKeySetRef keyset;
} PGPEventPassphraseData;
```

kPGPEvent_OutputEvent

If the initial call to `PGPDecode` did not include an output specification option, then this event will be sent whenever a new section of the message is encountered. This allows the application total flexibility in routing each output section.

If the initial call to `PGPDecode` did include an output specification option, then this event will not be sent and all output will go to the specified location. However, keys are handled as described in `kPGPEvent_KeyFoundEvent`.

The `messageType` indicates whether the section is text, data, or non-PGP. The `suggestedName` argument specifies the name the encrypted or signed file had when it was encrypted. The `forYourEyesOnly` flag is `TRUE` if the encryption specified the `PGPOForYourEyesOnly` option.

The event handler should use this information to specify a processing option appropriate for the output of the section. These options include:

- write the output to a file
- write the output to a buffer
- discard the output

The event handler should return an error if it cannot set an output option.

Data

```
typedef struct PGPEventOutputData_
{
    PGPUInt32 messageType;
    char *suggestedName;
    PGPBoolean forYourEyesOnly;
} PGPEventOutputData;
```

kPGPEvent_DecryptionEvent

Sent upon completion of the decode process to identify the symmetric (conventional) encryption algorithm used. This is primarily a debugging feature, since the actual selection depends upon both algorithm availability and user preferences (see `PGPOPreferredAlgorithms`).

Data

```
typedef struct PGPEventDecryptionData_
{
    PGPCipherAlgorithm
        algID;
} PGPEventDecryptionData;
```

kPGPEvent_EndLexEvent

Sent whenever a lexical section is completed (see the `BeginLexEvent` description for how sections are defined). The zero-based `sectionNumber` value indicates which section has been completed.

Data

```
typedef struct PGPEventEndLexData_
{
    PGPUInt32 sectionNumber;
} PGPEventEndLexData;
```

Public Key Encode and Decode Functions

PGPEncode

Encrypts a block of text according to the target context and specified options. This is **the** function for encrypting and signing data as PGP formatted output.

Syntax

```
PGPError PGPEncode(
                PGPContextRef pgpContext,
                PGPOptionListRef firstOption,
                ...,
                PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Allowed options include:

- One of `PGPOInputFile`, `PGPInputBuffer`, or `PGPOInputFileFSSpec` (required)
- One of `PGPOOutputFile`, `PGPOutputBuffer`, `PGPOAllocatedOutputBuffer`, `PGPODiscardOutput`, or `PGPOOutputFileFSSpec`
- `PGPOAppendOutput`
- `PGPOArmorOutput`
- `PGPOAskUserForEntropy`
- `PGPOCipherAlgorithm`
- `PGPOClearSign`
- `PGPOCommentString`
- `PGPOCompression`
- `PGPOConventionalEncrypt`
- `PGPODataIsASCII`
- `PGPODetachedSig`
- `PGPOEncryptToKey`
- `PGPOEncryptToKeySet`
- `PGPOEncryptToUserID`
- `PGPOEventHandler`
- `PGPOFailBelowValidity`
- `PGPOFileNameString`
- `PGPOForYourEyesOnly`
- `PGPOHashAlgorithm`
- `PGPOLocalEncoding`
- `PGPONullOption`
- `PGPOOmitMIMEVersion`
- `PGPOOutputLineEndType`
- `PGPOPasskeyBuffer`
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`
- `PGPOPGPMIMEEncoding`
- `PGPOPreferredAlgorithms`
- `PGPORawPGPInput`
- `PGPOSendNullEvents`
- `PGPOSignWithKey`
- `PGPOVersionString`
- `PGPOWarnBelowValidity`

Notes

See Chapter 3, "Option List Functions.", for a description of the PGPO option functions.

PGPDecode

Decrypts a block of text according to the target context and specified options. This is **the** function for decrypting and verifying PGP formatted data.

Syntax

```
PGPError PGPDecode(
            PGPContextRef pgpContext,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

pgpContext	the target context
firstOption	the initial option list instance
...	subsequent option list instances
PGPOLastOption()	must always appear as the final argument to terminate the argument list

Options

Allowed options include:

- One of PGPOInputFile, PGPInputBuffer, or PGPOInputFileFSSpec (required)
- One of PGPOOutputFile, PGPOutputBuffer, PGPOAllocatedOutputBuffer, PGPODiscardOutput, or PGPOOutputFileFSSpec
- PGPOAppendOutput
- PGPODetachedSig
- PGPOEventHandler
- PGPOFailBelowValidity
- PGPOImportKeysTo
- PGPOKeySetRef
- PGPOLocalEncoding
- PGPONullOption
- PGPOOutputLineEndType
- PGPOPasskeyBuffer
- PGPOPassphrase
- PGPOPassphraseBuffer

- PGPOPassThroughClearSigned
- PGPOPassThroughIfUnrecognized
- PGPOPassThroughKeys
- PGPORecursivelyDecode
- PGPOSendEventIfKeyFound
- PGPOSendNullEvents
- PGPOWarnBelowValidity

Notes

See Chapter 3, "Option List Functions.", for a description of the PGPO option functions.

Low-Level Cipher Functions - Hash

PGPNewHashContext

Creates a new hash context that utilizes the specified algorithm.

Syntax

```
PGPError PGPNewHashContext(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPHashAlgorithm algID,
             PGPHashContextRef *hashContext );
```

Parameters

pgpMemoryMgr	the target memory manager
algID	the hash algorithm to use
hashContext	the receiving field for the resultant hash context

PGPCopyHashContext

Creates an exact copy of the source hash context.

Syntax

```
PGPError PGPCopyHashContext(
             PGPHashContextRef hashContextOrig,
             PGPHashContextRef *hashContextCopy );
```

Parameters

hashContextOrig	the source hash context
hashContextCopy	the receiving field for the copy of the hash context

Notes

The caller is responsible for de-allocating the resultant hash context copy with PGPFreeHashContext.

PGPFreeHashContext

Frees the specified hash context.

Syntax

```
PGPError PGPFreeHashContext(
             PGPHashContextRef hashContext );
```

Parameters

`hashContext`	the target hash context

Notes

Hash contexts do *not* have associated reference counts – the context is always de-allocated.

PGPGetHashSize

Determines the resultant size of the associated hash in bytes, for example, a 160-bit hash may yield 20 bytes of resultant data.

Syntax

```
PGPError PGPGetHashSize(
             PGPHashContextRef hashContext,
             PGPSize *hashSize );
```

Parameters

`hashContext`	the target hash context
`hashSize`	the receiving field for the hash size (in bytes)

Notes

Used for generic code that may not know the size of the hash being produced.

PGPContinueHash

Continues the hash, accumulating an intermediate result.

Syntax

```
PGPError PGPContinueHash(
             PGPHashContextRef hashContext,
             const void *hashIn,
             PGPSize numBytes );
```

Parameters

`hashContext`	the target hash context
`hashIn`	the current hash data
`numBytes`	the length of the current hash data (in bytes)

Notes

Normally, `numBytes` should be passed as the value received from `PGPGetHashSize`.

PGPFinalizeHash

Finalizes the hash, placing the result into `hashOut`. The hash context is then automatically reset via `PGPResetHash`.

Syntax

```
PGPError PGPFinalizeHash(
             PGPHashContextRef hashContext,
             void *hashOut );
```

Parameters

`hashContext`	the target hash context
`hashOut`	the receiving buffer for the resultant hash data

Notes

Use `PGPGetHashSize` to ensure that the result buffer is of adequate size.

To obtain an intermediate result, use `PGPCopyHashContext` and then finalize the copy.

PGPResetHash

Resets a hash context as if it had been created anew. Any existing intermediate hash is lost.

Syntax

```
PGPError PGPResetHash( PGPHashContextRef hashContext );
```

Parameters

`hashContext`	the target hash context

Low-Level Cipher Functions - HMAC

PGPNewHMACContext

Creates a new hash context that utilizes the specified algorithm, and that is specifically intended for computing MAC (Message Authentication Code) values.

Syntax

```
PGPError PGPNewHMACContext(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPHashAlgorithm algID,
             PGPByte *secret,
             PGPSize secretLength,
             PGPHMACContextRef *hmacContext );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`algID`	the hash algorithm to use
`secret`	the MAC key for this HMAC context
`secretLength`	the length of the MAC key for this HMAC context (in bytes)
`hmacContext`	the receiving field for the resultant HMAC context

Notes

If `secret` is longer than the maximum HMAC block size (currently 64 bytes), then it is silently truncated.

PGPFreeHMACContext

Frees the specified HMAC context.

Syntax

```
PGPError PGPFreeHMACContext(
             PGPHMACContextRef hmacContext );
```

Parameters

`hmacContext`	the target HMAC context

Notes

HMAC contexts do *not* have associated reference counts – the context is always de-allocated.

PGPContinueHMAC

Continues the HMAC, accumulating an intermediate result.

Syntax

```
PGPError PGPContinueHMAC(
             PGPHMACContextRef hmacContext,
             const void *hmacIn,
             PGPSize numBytes );
```

Parameters

`hmacContext`	the target HMAC context
`hmacIn`	the current HMAC data
`numBytes`	the length of the current HMAC data

Notes

Normally, `numBytes` should be passed as the maximum HMAC blocksize (currently 64 bytes).

PGPFinalizeHMAC

Finalizes the HMAC, placing the result into `hmacOut`. The HMAC context is then automatically reset via `PGPResetHMAC`.

Syntax

```
PGPError PGPFinalizeHMAC(
             PGPHMACContextRef hmacContext,
             void *hmacOut );
```

Parameters

`hmacContext`	the target HMAC context
`hmacOut`	the receiving buffer for the resultant HMAC data

Notes

The result buffer should be at least the maximum HMAC block size (currently 64 bytes).

PGPResetHMAC

Resets an HMAC context as if it had been created anew. Any existing intermediate HMAC is lost.

Syntax

```
PGPError PGPResetHMAC( PGPHMACContextRef hmacContext );
```

Parameters

`hmacContext`	the target HMAC context

Low-Level Cipher Functions - Symmetric Cipher

PGPNewSymmetricCipherContext

Creates a new symmetric cipher based upon the specified algorithm.

Syntax

```
PGPError PGPNewSymmetricCipherContext(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPCipherAlgorithm algID,
             PGPSize keySize,
             PGPSymmetricCipherContextRef *cipherContext
             );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`algID`	the desired symmetric cipher algorithm
`keySize`	the desired key size (in bytes)
`cipherContext`	the receiving field for the resultant symmetric cipher context

Notes

Currently, all supported symmetric cipher algorithms have only one key size. Specifying the key size as `kPGPSymmetricCipherDefaultKeySize` will not only simplify coding, but also avoid errors. This is especially true if the PGPsdk developer avoids any specification of key size, and instead always obtains the effective key size from `PGPGetSymmetricCipherSizes`.

The resultant symmetric cipher context cannot be used until it has been initialized with `PGPInitSymmetricCipher`.

The caller is responsible for de-allocating the resultant symmetric cipher context with `PGPFreeSymmetricCipherContext` *unless* the copy is passed to a function that assumes ownership, for example `PGPNewCBCipherContext` or `PGPNewCFBCipherContext`.

PGPInitSymmetricCipher

Establishes the key for the symmetric cipher context.

Syntax

```
PGPError PGPInitSymmetricCipher(
             PGPSymmetricCipherContextRef cipherContext,
             const void *key );
```

Parameters

`cipherContext`	the target symmetric cipher context
`key`	the desired key

Notes

The key size is determined by the choice of symmetric cipher, and may be obtained with `PGPGetSymmetricCipherSizes`.

Since the key is copied into the symmetric cipher context and so is no longer needed, the caller is strongly encouraged to clear the key's memory upon successful return.

A symmetric cipher can be repeatedly reset and reused with different keys, which avoids having to create and destroy new contexts each time.

PGPCopySymmetricCipherContext

Creates an exact copy of the source symmetric cipher context, including its key.

Syntax

```
PGPError PGPCopySymmetricCipherContext(
             PGPSymmetricCipherContextRef
                cipherContextOrig,
             PGPSymmetricCipherContextRef
                *cipherContextCopy );
```

Parameters

`cipherContextOrig`	the source symmetric cipher context
`cipherContextCopy`	the receiving field for the copy of the symmetric cipher context

Notes

The caller is responsible for de-allocating the resultant symmetric cipher context copy with `PGPFreeSymmetricCipherContext` *unless* the copy is passed to a function that assumes ownership, for example `PGPNewCBCipherContext` or `PGPNewCFBCipherContext`.

PGPFreeSymmetricCipherContext

Frees the specified symmetric cipher context.

Syntax

```
PGPError PGPFreeSymmetricCipherContext(
             PGPSymmetricCipherContextRef
                cipherContext );
```

Parameters

`cipherContext`	the target symmetric cipher context

Notes

This function should only be called for those symmetric cipher contexts that are *not* passed to functions that assume ownership, for example `PGPNewCBCipherContext` or `PGPNewCFBCipherContext`.

Symmetric cipher contexts do *not* have associated reference counts – the context is always de-allocated.

Before de-allocating the context, the function erases all sensitive in-memory data.

PGPGetSymmetricCipherSizes

Returns the key and block sizes (in bytes) for the associated symmetric cipher.

Syntax

```
PGPError PGPGetSymmetricCipherSizes(
             PGPSymmetricCipherContextRef cipherContext,
             PGPSize *keySize,
             PGPSize *blockSize );
```

Parameters

`cipherContext`	the target symmetric cipher context
`keySize`	the receiving field for the associated cipher’s key size (in bytes)
`blockSize`	the receiving field for the associated cipher’s block size (in bytes)

PGPSymmetricCipherEncrypt

Encrypts one block of data, whose size is determined by the cipher (see `PGPGetSymmetricCipherSizes`).

Syntax

```
PGPError PGPSymmetricCipherEncrypt(
             PGPSymmetricCipherContextRef cipherContext,
             const void *plainText,
             void *cipherText );
```

Parameters

`cipherContext`	the target symmetric cipher context
`plainText`	the source buffer for the input **plain text**
`cipherText`	the receiving buffer for the output cipher text

Notes

This function should not be used to encrypt multiple blocks of data unless the key is changed for each block (usually through a chaining or feedback scheme), since it is considered bad cryptographic practice to reuse a key in a block cipher.

PGPSymmetricCipherDecrypt

Decrypts one block of data, whose size is determined by the target cipher context (see `PGPGetSymmetricCipherSizes`).

Syntax

```
PGPError PGPSymmetricCipherDecrypt(
             PGPSymmetricCipherContextRef cipherContext,
             const void *cipherText,
             void *plainText );
```

Parameters

`cipherContext`	the target symmetric cipher context
`cipherText`	the source buffer for the input cipher text
`plainText`	the receiving buffer for the output plain text

PGPWashSymmetricCipher

Hashes the current key of the specified symmetric cipher with the specified wash data to produce a new key.

Syntax

```
PGPError PGPWashSymmetricCipher(
             PGPSymmetricCipherContextRef cipherContext,
             void const *washData,
             PGPSize washLength );
```

Parameters

`cipherContext`	the target symmetric cipher context
`washData`	the wash data
`washLength`	the length of the wash data (in bytes)

PGPWipeSymmetricCipher

“Wipes” any sensitive data in the cipher. The cipher context remains “alive”, but its key must be reset before any more data can be encrypted.

Syntax

```
PGPError PGPWipeSymmetricCipher(
             PGPSymmetricCipherContextRef cipherContext );
```

Parameters

`cipherContext`	the target symmetric cipher context

Low-Level Cipher Functions - Cipher Block Chaining

PGPNewCBCContext

Creates a **cipher block chaining** context based upon the specified symmetric cipher.

Syntax

```
PGPError PGPNewCBCContext(
             PGPSymmetricCipherContextRef cipherContext,
             PGPCBCContextRef *chainingContext );
```

Parameters

`cipherContext`	the underlying symmetric cipher context
`chainingContext`	the receiving field for the resultant CBC context

Notes

A cipher block chaining context requires use of a symmetric cipher that has been created and whose key has been set. This key may be set explicitly with `PGPInitSymmetricCipher`, or set implicitly with `PGPInitCBC`.

Upon creation of the context, the `CBCRef` "owns" the symmetric `cipherContext` and will dispose of it properly (even if an error occurs). The caller should no longer reference it.

PGPInitCBC

Establishes the key and/or **intialization vector** for the cipher chaining context. One of `key` and `initVector` may be `NULL`, but not both.

Syntax

```
PGPError PGPInitCBC(
             PGPCBCContextRef chainingContext,
             const void *key,
             const void *initVector );
```

Parameters

`chainingContext`	the target CBC context
`key`	the desired key
`initVector`	the desired initialization vector data

Notes

The initialization vector (IV) size is assumed to be the same as the symmetric cipher block size.

Since both arguments are copied into the cipher chaining context, the caller is encouraged to clear their memory upon successful return.

Both `key` and `initializationVector` must be set prior to any cipher operations. However, as a convenience to the PGPsdk developer, these may be set in separate calls to `PGPInitCBC` and / or `PGPInitSymmetricCipher` since these values are commonly obtained from different sources at different times.

If the PGPsdk developer neglects to call `PGPInitCBC` to set the initialization vector, for example, always sets the key via `PGPInitSymmetricCipher`, then the initialization vector defaults to zeroes. Generally, it is better cryptographic practice to set the initialization vector to random data.

PGPCopyCBCContext

Creates an exact copy of the source chaining cipher context.

Syntax

```
PGPError PGPCopyCBCContext(
            PGPCBCContextRef chainingContextOrig,
            PGPCBCContextRef *chainingContextCopy );
```

Parameters

`chainingContextOrig`	the source CBC context
`chainingContextCopy`	the receiving field for the copy of the CBC context

Notes

The caller is responsible for de-allocating the resultant chaining cipher context copy with `PGPFreeCBCCipherContext`.

PGPFreeCBCContext

Decrements the reference count for the specified cipher block chaining context, and frees the context if the reference count reaches zero.

Syntax

```
PGPError PGPFreeCBCContext(
            PGPCBCContextRef chainingContext );
```

Parameters

`chainingContext`	the target cipher block chaining context

Notes

Before de-allocating the context, the function erases all associated in-memory data.

PGPCBCEncrypt

Encrypts the specified data according to the specified cipher block chaining context.

Syntax

```
PGPError PGPCBCEncrypt(
             PGPCBCContextRef chainingContext,
             const void *plainText,
             PGPSize plainTextLength,
             void *cipherText );
```

Parameters

`chainingContext`	the target CBC context
`plainText`	the data to encrypt
`plainTextLength`	the length of the data to encrypt (in bytes)
`cipherText`	the receiving buffer for the resultant encrypted data

Notes

Since cipher block chaining effectively changes the key for each block of plain text, `PGPCBCEncryp` can be called repeatedly to encrypt arbitrary amounts of data.

PGPCBCDecrypt

Decrypts the specified data according to the specified chaining context.

Syntax

```
PGPError PGPCBCDecrypt(
             PGPCBCContextRef chainingContext,
             const void *cipherText,
             PGPSize cipherTextLength,
             void *plainText );
```

Parameters

`chainingContext`	the target CBC context
`cipherText`	the data to decrypt
`cipherTextLength`	the length of the data to decrypt (in bytes)
`plainText`	the receiving buffer for the resultant plain text

PGPCBCGetSymmetricCipher

Get the symmetric cipher context being used by the specified cipher block chaining context.

Syntax

```
PGPError PGPCBCGetSymmetricCipher(
             PGPCBCContextRef chainingContext,
             PGPSymmetricCipherContextRef
                *cipherContext );
```

Parameters

`chainingContext`	the target CBC context
`cipherContext`	the receiving field for the symmetric cipher context

Notes

`cipherContext` is the actual `PGPSymmetricCipherContext`, and *not* a copy. Since the chaining context "owns" the symmetric cipher, the caller may copy the symmetric cipher, but should neither free nor de-reference it.

Once obtained, the symmetric cipher reference can be used to obtain attributes of the underlying cipher, for example, its block size.

Low-Level Cipher Functions - Cipher Feedback Block

PGPNewCFBContext

Creates a new feedback context based upon the specified symmetric cipher. The specified interleave factor determines the number of cipher blocks through which the feedback mechanism will cycle.

Syntax

```
PGPError PGPNewCFBContext(
             PGPSymmetricCipherContextRef cipherContext,
             PGPUInt16 interleaveFactor,
             PGPCFBContextRef *feedbackContext );
```

Parameters

`cipherContext`	the underlying symmetric cipher context
`interleaveFactor`	the desired number of cipher blocks in the feedback

	loop
`feedbackContext`	the receiving field for the resultant CFB context

Notes

A cipher feedback context requires use of a symmetric cipher that has been created and whose key has been set. This key may be set explicitly with `PGPInitSymmetricCipher`, or set implicitly with `PGPInitCFB`.

After the call, the `CFBRef` "owns" the symmetric `cipherContext` and will dispose of it properly (even if an error occurs). The caller should no longer reference it.

The choice of interleave factor affects the size of the resultant feedback context, but does not affect its performance. However, while the PGPsdk API currently supports interleaving, it is not yet fully implemented. As such, the interleave factor should always be specified as one.

PGPInitCFB

Establishes the key(s) and / or initialization vector(s) for the cipher feedback context. One of `key` and `initializationVector` may be `NULL`, but not both.

Syntax

```
PGPError PGPInitCFB(
            PGPCFBContextRef feedbackContext,
            const void *key,
            const void *initVector );
```

Parameters

`feedbackContext`	the target CFB context
`key`	the desired key data
`initVector`	the desired initialization vector data

Notes

The key data size is assumed to be the key size of the associated symmetric cipher, times the feedback context's interleave factor; the initialization vector (IV) data size is assumed to be the block size of the associated symmetric cipher, times the feedback context's interleave factor.

Since both arguments are copied into the cipher feedback context, the caller is encouraged to clear their memory upon successful return.

Both `key` and `initializationVector` must be set prior to any cipher operations. However, as a convenience to the PGPsdk developer, these may be set in separate calls to `PGPInitCFB` and / or `PGPInitSymmetricCipher` since these values are commonly obtained from different sources at different times.

If the PGPsdk developer neglects to call `PGPInitCFB` to set the initialization vector, for example, always sets the key via `PGPInitSymmetricCipher`, then the initialization vector defaults to zeroes. Generally, it is better cryptographic practice to set the initialization vector to random data.

PGPCopyCFBContext

Creates an exact copy of the source feedback cipher context.

Syntax

```
PGPError PGPCopyCFBContext(
             PGPCFBContextRef feebackContextOrig,
             PGPCFBContextRef *feebackContextCopy );
```

Parameters

`feebackContextOrig`	the source CFB context
`feebackContextCopy`	the receiving field for the copy of the CFB context

Notes

The caller is responsible for de-allocating the resultant feedback cipher context copy with `PGPFreeCFBCipherContext`.

PGPFreeCFBContext

Decrements the reference count of the specified cipher feedback context, and frees the context if the reference count reaches zero.

Syntax

```
PGPError PGPFreeCFBContext(
             PGPCFBContextRef feedbackContext );
```

Parameters

`feedbackContext`	the target cipher feedback context

Notes

Before de-allocating the context, the function erases all associated in-memory data.

PGPCFBEncrypt

Encrypts the specified data according to the specified feedback context.

Syntax

```
PGPError PGPCFBEncrypt(
             PGPCFBContextRef feedbackContext,
             const void *plainText,
             PGPSize plainTextLength,
             void *cipherText );
```

Parameters

`feedbackContext`	the target CFB context
`plainText`	the data to encrypt
`plainTextLength`	the length of the data to encrypt (in bytes)
`cipherText`	the receiving buffer for the resultant encrypted data

Notes

Call repeatedly to encrypt arbitrary amounts of data.

PGPCFBDecrypt

Decrypts the specified data according to the specified feedback context.

Syntax

```
PGPError PGPCFBDecrypt(
             PGPCFBContextRef feedbackContext,
             const void *cipherText,
             PGPSize cipherTextLength,
             void *plainText );
```

Parameters

`feedbackContext`	the target CFB context
`cipherText`	the data to decrypt
`cipherTextLength`	the length of the data to decrypt (in bytes)
`plainText`	the receiving buffer for the resultant plain text

PGPCFBGetSymmetricCipher

Get the symmetric cipher context associated with the specified cipher feedback context.

Syntax

```
PGPError PGPCFBGetSymmetricCipher(
             PGPCFBContextRef feedbackContext,
             PGPSymmetricCipherContextRef
                *cipherContext );
```

Parameters

`feedbackContext`	the target CFB context
`cipherContext`	the receiving field for the context of the associated symmetric cipher

Notes

`cipherContext` is the actual `PGPSymmetricCipherContext`, and *not* a copy. Since the feedback context "owns" the symmetric cipher, the caller should neither free nor de-reference it, but may copy it.

Once obtained, the symmetric cipher reference can be used to obtain attributes of the underlying cipher, for example, its block size.

PGPCFBGetRandom

Fetches pseudo-random bytes from the specified cipher feedback context up to a maximum of `requestCount` bytes, and indicates the actual number of pseudo-random bytes obtained.

Syntax

```
PGPError PGPCFBGetRandom(
             PGPCFBContextRef feedbackContext,
             PGPSize requestCount,
             void *randomData,
             PGPSize *randomDataCount );
```

Parameters

`feedbackContext`	the target CFB context
`requestCount`	the maximum number of pseudo-random bytes to

	fetch
`randomData`	the receiving buffer for the pseudo-random bytes
`randomDataCount`	the receiving field for the actual number of pseudo-random bytes fetched

Notes

The receiving buffer must be at least `requestCount` bytes in length.

PGPCFBRandomCycle

Makes more pseudo-random bytes available by iterating through the existing random number pool, and applying the supplied **salt**.

Syntax

```
PGPError PGPCFBRandomCycle(
             PGPCFBContextRef feedbackContext,
             const void *salt );
```

Parameters

`feedbackContext`	the target CFB context
`salt`	the additional random byte data

Notes

The number of salt bytes is assumed to equal the block size of the associated symmetric cipher.

PGPCFBRandomWash

Hashes the associated specified symmetric cipher's key and initialization vector with the specified wash data to produce a new key and a new initialization vector.

Syntax

```
PGPError PGPCFBRandomWash(
             PGPCFBContextRef feedbackContext,
             const void *washData,
             PGPSize washDataLength );
```

Parameters

`feedbackContext`	the target CFB context
`washData`	the wash data
`washDataLength`	the length of the wash data (in bytes)

Notes

If `washDataLength` is less than the symmetric cipher block size, then padding bytes are used. If `washDataLength` is greater than the symmetric cipher block size, then multiple iterations occur. Passing "extra" wash data never reduces the resultant cryptographic strength of the resultant cipher text, and often increases it.

PGPCFBSync

Reset the feedback mechanism to use the currently available data plus an additional number of previous bytes, such that the resultant data length equals the cipher block size.

Syntax

```
PGPError PGPCFBSync( PGPCFBContextRef feedbackContext );
```

Parameters

`feedbackContext`	the target CFB context

Notes

This effectively changes the cipher block boundary.

Low-Level Cipher Functions - Public Key

PGPNewPublicKeyContext

Creates a context for public key operations based on the specified key and using the specified message format.

Syntax

```
PGPError PGPNewPublicKeyContext(
             PGPKeyRef key,
             PGPPublicKeyMessageFormat messageFormat,
             PGPPublicKeyContextRef *publicKeyContext );
```

Parameters

`key`	the target key
`messageFormat`	the desired message format
`publicKeyContext`	the receiving field for the resultant public key context

PGPFreePublicKeyContext

Decrements the reference count of the specified public key context, and frees the context if the reference count reaches zero.

Syntax

```
PGPError PGPFreePublicKeyContext(
             PGPPublicKeyContextRef
             publicKeyContext );
```

Parameters

`publicKeyContext`	the target public key context

PGPGetPublicKeyOperationSizes

Returns the sizes associated with the specified public key context. A resultant value of zero indicates that the associated operation is not available, for example if `maxSignatureSize` is zero, then signing is not a supported operation.

Syntax

```
PGPError PGPGetPublicKeyOperationSizes(
             PGPPublicKeyContextRef publicKeyContext,
             PGPSize *maxDecryptedBufferSize,
             PGPSize *maxEncryptedBufferSize,
             PGPSize *maxSignatureSize );
```

Parameters

`publicKeyContext`	the target public key context
`maxDecryptedBufferSize`	the receiving field for the decryption buffer size (in bytes)
`maxEncryptedBufferSize`	the receiving field for the encryption buffer size (in bytes)
`maxSignatureSize`	the receiving field for the signature size (in bytes)

PGPPublicKeyEncrypt

Encrypts one block of data.

Syntax

```
PGPError PGPPublicKeyEncrypt(
             PGPPublicKeyContextRef publicKeyContext,
             void const *plainText,
             PGPSize plainTextLength,
             void *cipherText,
             PGPSize *cipherTextLength );
```

Parameters

`publicKeyContext`	the target public key context
`plainText`	the buffer containing the input plain text
`plainTextLength`	the length of the input plain text (in bytes)
`cipherText`	the receiving buffer for the output cipher text, which must be at least `maxEncryptedBufferSize` (obtained from `PGPGetPublicKeyOperationSizes`)
`cipherTextLength`	the receiving field for the resultant length of the output cipher text (in bytes)

PGPPublicKeyVerifySignature

Verifies a signature on a message hash, which is both finalized and freed. A return value of `kPGPError_NoErr` indicates a successful verification.

Syntax

```
PGPError PGPPublicKeyVerifySignature(
             PGPPublicKeyContextRef publicKeyContext,
             PGPHashContextRef hashContext,
             void const *signature,
             PGPSize signatureSize );
```

Parameters

`publicKeyContext`	the target public key context
`hashContext`	the target hash context
`signature`	the target signature
`signatureSize`	the length of the target signature (in bytes)

Notes

The message hash should *not* have been finalized prior to calling this function.

PGPPublicKeyVerifyRaw

Verifies a signature on raw, signed data in a low-level buffer. A return value of `kPGPError_NoErr` indicates a successful verification.

Syntax

```
PGPError PGPPublicKeyVerifyRaw(
             PGPPublicKeyContextRef publicKeyContext,
             void const *signedData,
             PGPSize signedDataSize,
             void const *signature,
             PGPSize signatureSize );
```

Parameters

`publicKeyContext`	the target public key context
`signedData`	the target signed data
`signedDataSize`	the length of the target signed data (in bytes)
`signature`	the target signature
`signatureSize`	the length of the target signature (in bytes)

Notes

This function will fail if the target public context is of type `kPGPPublicKeyMessageFormat_PGP`.

Low-Level Cipher Functions - Private Key

PGPNewPrivateKeyContext

Creates a context for private key operations based on the specified key and using the specified message format.

Syntax

```
PGPError PGPNewPrivateKeyContext(
            PGPKeyRef key,
            PGPPrivateKeyMessageFormat messageFormat,
            PGPPrivateKeyContextRef *privateKeyContext,
            PGPOptionListRef passphraseOption,
            PGPOLastOption() );
```

Parameters

`key`	the target key, which must be a public/private key pair
`messageFormat`	the desired message format
`privateKeyContext`	the receiving field for the resultant private key context
`passphraseOption`	passphrase or passkey which unlocks the private key
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

The `passphraseOption` must be one of the following:

- `PGPOPasskeyBuffer`
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`

Notes

The `passphraseOption` is required, not optional.

PGPFreePrivateKeyContext

Decrements the reference count of the specified private key context, and frees the context if the reference count reaches zero.

Syntax

```
PGPError PGPFreePrivateKeyContext(
            PGPPrivateKeyContextRef privateKeyContext );
```

Parameters

`privateKeyContext`	the target `private` key context

Notes

Before de-allocating the context, the function erases all sensitive in-memory data.

PGPGetPrivateKeyOperationSizes

Returns the sizes associated with the specified private key context. A resultant value of zero indicates that the associated operation is not available.

Syntax

```
PGPError PGPGetPrivateKeyOperationSizes(
             PGPPrivateKeyContextRef privateKeyContext,
             PGPSize *maxDecryptedBufferSize,
             PGPSize *maxEncryptedBufferSize,
             PGPSize *maxSignatureSize );
```

Parameters

`privateKeyContext`	the target private key context
`maxDecryptedBufferSize`	the receiving field for the decryption buffer size (in bytes)
`maxEncryptedBufferSize`	the receiving field for the encryption buffer size (in bytes)
`maxSignatureSize`	the receiving field for the signature size (in bytes)

PGPPrivateKeyDecrypt

Decrypts one block of data.

Syntax

```
PGPError PGPPrivateKeyDecrypt(
             PGPPrivateKeyContextRef privateKeyContext,
             void const *cipherText,
             PGPSize cipherTextLength,
             void *plainText,
             PGPSize *plainTextLength );
```

Parameters

`privateKeyContext`	the target private key context
`cipherText`	the buffer containing the input cipher text
`cipherTextLength`	the length of the input cipher text (in bytes)
`plainText`	the receiving buffer for the output plain text, which must be at least `maxDecryptedBufferSize` (obtained from `PGPGetPrivateKeyOperationSizes`)
`plainTextLength`	the receiving field for the resultant length of the output plain text

PGPPrivateKeySign

Signs a message hash according to the specified private key context, yielding the signature and its length (in bytes). The target hash context is both finalized and freed.

Syntax

```
PGPError PGPPrivateKeySign(
             PGPPrivateKeyContextRef privateKeyContext,
             PGPHashContextRef hashContext,
             void *signature,
             PGPSize *signatureSize );
```

Parameters

`privateKeyContext`	the target private key context
`hashContext`	the target hash context
`signature`	the receiving field for the signature, which must be at least `maxSignatureSize` (obtained from `PGPGetPrivateKeyOperationSizes`)
`signatureSize`	the receiving field for the resultant length of the signature (in bytes)

Notes

The message hash should *not* have been finalized prior to calling this function.

PGPPrivateKeySignRaw

Signs raw data in a low-level buffer according to the specified private key context, yielding the signature and its length (in bytes).

Syntax

```
PGPError PGPPrivateKeySignRaw(
             PGPPrivateKeyContextRef privateKeyContext,
             void const *signedData,
             PGPSize signedDataSize,
             void const *signature,
             PGPSize *signatureSize );
```

Parameters

`privateKeyContext`	the target private key context
`signedData`	the target signed data
`signedDataSize`	the length of the target signed data(in bytes)
`signature`	the target signature
`signatureSize`	the length of the target signature (in bytes)

Low-Level Cipher Functions - Misc.

PGPDiscreteLogExponentBits

For a given prime modulus size (in bits), this function determines an appropriate exponent size (in bits) such that the work factor required to find a discrete log modulo the modulus is approximately equal to half the length of the exponent.

Syntax

```
PGPError PGPDiscreteLogExponentBits(
             PGPUInt32 modulusBits,
             PGPUInt32 *exponentBits );
```

Parameters

`modulusBits`	the size of a prime modulus (in bits)
`exponentBits`	the resultant appropriate number of exponent bits

Notes

The resultant exponent size may be used directly as the size of a sub-group in a discrete log signature scheme, but should be increased by 50% for encryption schemes.

Feature (Capability) Query Functions 6

Introduction

When one considers the present state of U.S. export law and the continuously evolving set of cryptographic standards, algorithms, and formats, the simultaneous existence of multiple versions of the PGPsdk becomes a very real possibility. For example, one instance of the PGPsdk library may support encryption, while another supports signing but not encryption. By including functions that return version numbers and the availability of specific features (capabilities), the PGPsdk provides applications with a measure of version independence, as well as a specific and extensible mechanism for determining feature availability.

The feature query functions that allow the caller to determine the availability of a specific feature before attempting to use it are the only supported means for determining such availability. The PGPsdk version number should *not* be used to determine feature availability. As the PGPsdk library evolves and adopts a more customized, modular build model that may include “stub” functions that do nothing except return an appropriate error code, the presence and use of these feature query functions can only increase in importance.

Header Files

`pgpFeatures.h`

Feature (Capability) Query Functions

PGPGetFeatureFlags

Retrieves the flags associated with the specified feature selector. A return value of `kPGPError_ItemNotFound` indicates that the `featureSelector` value was not recognized.

Syntax

```
PGPError PGPGetFeatureFlags(
            PGPFeatureSelector featureSelector,
            PGPFlags *featureFlags );
```

Parameters

`featureSelector` the feature flags to obtain, which recognizes

	`kPGPFeatures_…Selector` values
`featureFlags`	the receiving field for the feature flags

Notes

Since `flags` is an encoded value, individual features should always be extracted by presenting the `PGPFeatureExists` macro (defined in `pgpFeatures.h`) with the appropriate `kPGPFeatureMask_…` value.

PGPCountPublicKeyAlgorithms

Provides the number of available public key algorithms.

Syntax

```
PGPError PGPCountPublicKeyAlgorithms(
            PGPUInt32 *numPKAlgs );
```

Parameters

`numPKAlgs`	the receiving field for the number of available public key algorithms

Notes

Use this count as the exclusive upper limit when indexing through the available algorithms.

PGPGetIndexedPublicKeyAlgorithmInfo

Provides a means of indexing through the available public key algorithms and accessing their associated information, which is of type `PGPPublicKeyAlgorithmInfo`.

Syntax

```
PGPError PGPGetIndexedPublicKeyAlgorithmInfo(
            PGPUInt32 index,
            PGPPublicKeyAlgorithmInfo *info );
```

Parameters

`index`	the index (zero-based) of the desired public key algorithm
`info`	the receiving field for the associated algorithm information

PGPCountSymmetricCiphers

Provides the number of available symmetric ciphers.

Syntax

```
PGPError PGPCountSymmetricCiphers(
            PGPUInt32 *numSymmetricCiphers );
```

Parameters

`numSymmetricCiphers`	the receiving field for the number of available symmetric ciphers

Notes

Use this count as the exclusive upper limit when indexing through the available symmetric ciphers (see the sample code for `PGPGetIndexedSymmetricCipherInfo`).

PGPGetIndexedSymmetricCipherInfo

Provides a means of indexing through the available symmetric ciphers and accessing the associated information, which is of type `PGPSymmetricCipherInfo`.

Syntax

```
PGPError PGPGetIndexedSymmetricCipherInfo(
             PGPUInt32 index,
             PGPSymmetricCipherInfo *info );
```

Parameters

`index`	the index (zero-based) of the desired symmetric cipher
`info`	the receiving field for the associated information

PGPGetSDKVersion

Places the PGPsdk API version number into the referenced field. Since the version number is encoded, its components should always be extracted using the `PGPMajorVersion`, `PGPMinorVersion`, and `PGPRevVersion` macros defined in `pgpUtilities.h`.

Syntax

```
PGPError PGPGetSDKVersion( PGPUInt32 *version );
```

Parameters

`version`	the receiving field for the version number value

Notes

The version number reflects the API version, and not the release version of the packaged software developer’s kit. Generally speaking, the API version is independent of the version number reported by the PGPsdk.

PGPGetSDKString

A convenience function that yields a *C* language string of the form:

```
PGPsdk Version Version 1.5 (C) 1997-1998 Network Associates, Inc.
```

This function is similar of the sample code included for `PGPGetSDKVersion`, except for the fact that that it does not include the revision number.

Syntax

```
PGPError PGPGetSDKString( char theString[ 256 ] );
```

Parameters

`theString[ 256 ]`	a buffer having a minimum length of 256 bytes to receive the PGPsdk API version string

Utility Toolbox 7

Introduction

The PGPsdk includes miscellaneous utility functions that relate to multiple functional areas, such as:

- memory manager creation and management
- context creation and management
- file specification
- preferences
- date/time
- network library management
- error code to error string conversion

Header Files

`pgpMemoryMgr.h`

`pgpPubTypes.h`

`pgpSDKPrefs.h`

`pgpUtilities.h`

PGPsdk Management Functions

PGPsdkInit

Initializes the PGPsdk global state. *This function must be called prior to using any part of the PGPsdk.*

Syntax

```
PGPError PGPsdkInit( void );
```

Notes

Multiple calls to this function will *not* re-initialize the global variables. Instead, a mechanism similar to the opaque data type reference count mechanism tracks the calls. This frees the PGPsdk developer from having to worry about whether or not the global state has already been initialized, since a subsequent initialization will not adversely affect the global state.

The caller is responsible for freeing any and all resources held by the PGPsdk with `PGPsdkCleanup`.

PGPsdkCleanup

Releases any and all resources held by the PGPsdk.

Syntax

```
PGPError PGPsdkCleanup( void );
```

Notes

This function should be called only after freeing the last `PGPContext`. Any subsequent usage of the PGPsdk must first call `PGPsdkInit`.

Memory Manager Creation and Management Functions

PGPNewMemoryMgr

Creates a memory manager that employs the default PGPsdk memory management functions.

Syntax

```
PGPError PGPNewMemoryMgr(
            PGPFlags reserved,
            PGPMemoryMgrRef *pgpMemoryMgr );
```

Parameters

`reserved`	reserved flags; must be zeroes
`pgpMemoryMgr`	the receiving field for the new memory manager

PGPNewMemoryMgrCustom

Creates a PGPMemoryMgr that employs user-defined memory management functions.

Syntax

```
PGPError PGPNewMemoryMgrCustom(
             PGPNewMemoryMgrStruct const
                *pgpMemoryMgrData,
             PGPMemoryMgrRef *pgpMemoryMgr );
```

Parameters

pgpMemoryMgrData	the custom memory management information
pgpMemoryMgr	the receiving field for the new memory manager

Notes

The PGPNewMemoryMgrStruct member sizeofStruct *must* be specified as the special value sizeof(PGPNewMemoryMgrStruct).

PGPFreeMemoryMgr

Decrements the reference count for the specified memory manager (created by either PGPNewMemoryMgr or PGPNewMemoryMgrCustom), and frees the memory manager if the reference count reaches zero.

Syntax

```
PGPError PGPFreeMemoryMgr( PGPMemoryMgrRef pgpMemoryMgr );
```

Parameters

pgpMemoryMgr	the target memory manager

Notes

A PGPMemoryMgr must *not* be freed until and unless all data items allocated using that memory manager have been explicitly freed.

PGPMemoryMgrIsValid

Returns TRUE if the target memory manager is non-NULL and references a bona fide memory manager.

Syntax

```
PGPBoolean PGPMemoryMgrIsValid(
             PGPMemoryMgrRef pgpMemoryMgr );
```

Parameters

`pgpMemoryMgr`	the target memory manager

PGPSetDefaultMemoryMgr

Whereas most PGPsdk functions require a `context` parameter (which contains an embedded PGP memory manager context), some PGPsdk functions don't require a `context` parameter and thus don't specify what memory manager to use. This function, `PGPSetDefaultMemoryMgr()`, determines which memory manager the PGPsdk will use in such situations. To obtain the current value of the default memory manager, use `PGPGetDefaultMemoryMgr()`.

Syntax

```
PGPError PGPSetDefaultMemoryMgr(
             PGPMemoryMgrRef pgpMemoryMgr );
```

Parameters

`pgpMemoryMgr`	the target memory manager

PGPGetDefaultMemoryMgr

Returns the current value of the default memory manager. If the client code has not already set the default memory manager via `PGPSetDefaultMemoryManager()`, then a new memory manager is created using `PGPNewMemoryMgr()`, and that value is both set as the new global memory manager and returned as the function result.

Syntax

```
PGPMemoryMgrRef PGPGetDefaultMemoryMgr( void );
```

Notes

Whereas most PGPsdk functions require a `context` parameter (which contains an embedded PGP memory manager context), some PGPsdk functions don't require a `context` parameter and thus don't specify what memory manager to use. The PGPsdk uses the default memory manager in such situations.

PGPSetMemoryMgrCustomValue

Sets the user-defined data associated with the specified memory manager to that specified by `userValue`.

Syntax

```
PGPError PGPSetMemoryMgrCustomValue(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPUserValue userValue );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`userValue`	the associated (replacement) user-defined data

PGPGetMemoryMgrCustomValue

Retrieves the user-defined data associated with the specified memory manager.

Syntax

```
PGPError PGPGetMemoryMgrCustomValue(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPUserValue *userValue );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`userValue`	the receiving field for the associated user-defined data

PGPGetMemoryMgrDataInfo

Returns a `PGPFlags` value indicating the validity and security of the target memory block, as well as whether or not that block can be paged.

Syntax

```
PGPFlags PGPGetMemoryMgrDataInfo( void *allocation );
```

Parameters

`allocation`	the target memory block

PGPNewData

Allocates the specified number of 8-bit bytes of memory, using the memory allocation function associated with the specified memory manager. If the `flags` argument is specified as `kPGPMemoryMgrFlags_Clear`, then the resultant memory will be initialized to zeroes, overriding any custom setting.

Syntax

```
void *PGPNewData(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPSize allocationSize,
             PGPMemoryMgrFlags flags );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`allocationSize`	the number of 8-bits bytes to be allocated
`flags`	the desired memory manager flags

Notes

`PGPNewData` is used internally by the PGPsdk `PGPNew`... functions. Client code should rarely, if ever, have a reason to use this function.

Memory allocated with `PGPNewData` should always be de-allocated with `PGPFreeData`.

A return value of `NULL` indicates failure.

PGPNewSecureData

Allocates the specified number of 8-bit bytes of memory, using the memory allocation function associated with the specified memory manager. The allocated memory is intended to store sensitive data such as passphrases, and so:

- the function attempts to preclude the allocated memory from being swapped to secondary storage, thus facilitating later clearing of that memory
- `PGPFreeData` automatically clears memory allocated with this function prior to its being de-allocated

If the `flags` argument is specified as `kPGPMemoryMgrFlags_Clear`, then the resultant memory will be initialized to zeroes at allocation time, overriding any custom setting.

Syntax

```
void *PGPNewSecureData(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPSize allocationSize,
             PGPMemoryMgrFlags flags );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`allocationSize`	the number of 8-bit bytes to be allocated
`flags`	the desired memory manager flags

Notes

Memory allocated with `PGPNewSecureData` should always be de-allocated with `PGPFreeData`.

A return value of `NULL` indicates failure.

Not all platforms support page locking or other similar mechanism. Those that do may restrict it to certain classes of users, for example, the superuser. Still, the PGPsdk utilizes whatever facilities do exist for the platform, and ensures erasure of the *resident* memory upon de-allocation.

PGPReallocData

Re-allocates the specified number of 8-bit bytes of memory, using the memory re-allocation function associated with the specified memory manager.t

Syntax

```
PGPError PGPReallocData(
            PGPMemoryMgrRef pgpMemoryMgr,
            void **allocation,
            PGPSize newAllocationSize,
            PGPMemoryMgrFlags flags );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`allocation`	the target memory block, which is also the receiving field for the pointer to the re-allocated memory.
`newAllocationSize`	the number of 8-bit bytes to be allocated
`flags`	the desired memory manager flags

Notes

Memory re-allocated with `PGPReallocData` should always be de-allocated with `PGPFreeData`.

If allocation is specified as `NULL` , then the function simply allocates a new memory block having the specified size (see `PGPNewData`).

If the `flags` argument is specified as `kPGPMemoryMgrFlags_Clear`, then the resultant re-allocated memory will be initialized to zeroes, overriding any custom setting.

The resultant re-allocation is *not* guaranteed to start at the same address, even when `newAllocationSize` is smaller than the original size.

PGPFreeData

Frees memory allocated with `PGPNewData` and `PGPNewSecureData`. Memory allocated with `PGPNewSecureData` is cleared prior to its being freed.

Syntax

```
PGPError PGPFreeData( void *allocation );
```

Parameters

`allocation`	the target data in memory

Notes

The operation will fail silently if `allocation` is `NULL`, or if the associated internal header control block is corrupted.

Context Creation and Management Functions

PGPNewContext

Creates a context that employs the default PGPsdk memory management functions.

Syntax

```
PGPError PGPNewContext(
            PGPUInt32 clientAPIVersion,
            PGPContextRef *pgpContext );
```

Parameters

`clientAPIVersion`	the version of the current PGPsdk client API
`pgpContext`	the receiving field for the new context

Notes

`clientAPIVersion` should always be specified as the special value `kPGPsdkAPIVersion`.

PGPNewContextCustom

Creates a `PGPContext` that employs the memory management functions defined by the `memoryMgr` member of the `pgpContextStruct` argument. The custom information is passed as a `PGPNewContextStruct`, which may include a custom memory manager (see `PGPNewMemoryMgr` and `PGPNewMemoryMgrCustom`).

Syntax

```
PGPError PGPNewContextCustom(
             PGPUInt32 clientAPIVersion,
             PGPNewContextStruct const *pgpCustomData,
             PGPContextRef *pgpContext );
```

Parameters

`clientAPIVersion`	the version of the current PGPsdk client API
`pgpCustomData`	the custom context information
`pgpContext`	the receiving field for the new context

Notes

`clientAPIVersion` should always be specified as the special value `kPGPsdkAPIVersion`.

The `PGPNewContextStruct` member `sizeofStruct` *must* be specified as the special value `sizeof( PGPNewContextStruct )`.

PGPFreeContext

Decrements the reference count for the specified context (created by either `PGPNewContext` or `PGPNewContextCustom`), and frees the context if the reference count reaches zero.

Syntax

```
PGPError PGPFreeContext( PGPContextRef pgpContext );
```

Parameters

`pgpContext`	the target context

Notes

A `PGPContext` must *not* be freed until and unless all data items allocated using that context have been explicitly freed.

PGPSetContextUserValue

Sets the user-defined data associated with the specified context to that specified by `userValue`.

Syntax

```
PGPError PGPSetContextUserValue(
             PGPContextRef pgpContext,
             PGPUserValue userValue );
```

Parameters

`pgpContext`	the target context
`userValue`	the associated (replacement) user-defined data

PGPGetContextMemoryMgr

Returns the memory manager associated with the specified context.

Syntax

```
PGPMemoryMgrRef PGPGetContextMemoryMgr(
            PGPContextRef pgpContext );
```

Parameters

`pgpContext`	the target context

PGPContextGetRandomBytes

Places the pseudo-random bytes associated with the specified context into the specified buffer. A maximum of `availLength` bytes is retrieved. The function returns `kPGPError_OutOfEntropy` if the specified context's global random pool does not have sufficient entropy.

Syntax

```
PGPError PGPContextGetRandomBytes(
            PGPContextRef pgpContext,
            void *dataBuf,
            PGPSize availLength );
```

Parameters

`pgpContext`	the target context
`dataBuf`	the receiving buffer for the associated pseudo-random bytes
`availLength`	the length of the receiving buffer

Notes

The size of the global random pool and its entropy are independent of one another.

PGPGetContextUserValue

Retrieves the user-defined data associated with the specified context.

Syntax

```
PGPError PGPGetContextUserValue(
            PGPContextRef pgpContext,
            PGPUserValue *userValue );
```

Parameters

`pgpContext`	the target context
`userValue`	the receiving field for the associated user-defined data

File Specification Functions

PGPNewFileSpecFromFSSpec (*MacOS platforms only*)

Creates a file specification from the specified Macintosh FS specification.

Syntax

```
PGPError PGPNewFileSpecFromFSSpec(
             PGPContextRef pgpContext,
             const FSSpec *spec,
             PGPFileSpecRef *fileRef );
```

Parameters

`pgpContext`	the target context
`spec`	the source Macintosh FS specification
`fileRef`	the receiving field for the resultant file specification

Notes

The caller is responsible for de-allocating the resultant file specification with `PGPFreeFileSpec`.

PGPNewFileSpecFromFullPath (*Non-MacOS platforms only*)

Creates a file specification from a pathname.

Syntax

```
PGPError PGPNewFileSpecFromFullPath(
             PGPContextRef pgpContext,
             char const *pathname,
             PGPFileSpecRef *fileRef );
```

Parameters

`pgpContext`	the target context
`pathname`	the source pathname
`fileRef`	the receiving field for the resultant file specification

Notes

The caller is responsible for de-allocating the resultant file specification with `PGPFreeFileSpec`.

PGPCopyFileSpec

Creates an exact copy of a PGPFileSpecRef.

Syntax

```
PGPError PGPCopyFileSpec(
             PGPFileSpecRef fileSpecOrig,
             PGPFileSpecRef *fileSpecCopy );
```

Parameters

`fileSpecOrig`	the source file specification
`fileSpecCopy`	the receiving field for the copy of the file specification

Notes

The caller is responsible for de-allocating the resultant file specification copy with `PGPFreeFileSpec`.

PGPFreeFileSpec

Decrements the reference count for the specified file specification, and frees the file specification if the reference count reaches zero.

Syntax

```
PGPError PGPFreeFileSpec(
             PGPFileSpecRef fileSpecRef );
```

Parameters

`fileSpecRef`	the target file specification

PGPGetFSSpecFromFileSpec (*MacOS platforms only*)

Converts the specified file specification to a Macintosh FS specification.

Syntax

```
PGPError PGPGetFSSpecFromFileSpec(
             PGPFileSpecRef fileSpec,
             FSSpec *fsSpec );
```

Parameters

`fileSpec`	the source file specification
`fsSpec`	the receiving field for the resultant Macintosh FS specification

PGPGetFullPathFromFileSpec *(Non-MacOS platforms only)*

Converts the specified file specification to a file pathname, and places it into dynamically allocated memory.

Syntax

```
PGPError PGPGetFullPathFromFileSpec(
             PGPFileSpecRef fileSpec,
             char **fullPathPtr );
```

Parameters

`fileSpec`	the target file specification
`fullPathPtr`	the receiving field for a pointer to the resultant full pathname

Notes

The caller is responsible for de-allocating the resultant pathname with `PGPFreeData`.

PGPMacBinaryToLocal *(MacOS platforms only)*

Converts a MacOS MacBinary file to files containing its data fork and resource fork. The source file is deleted upon conversion.

A return value of `kPGPError_NoMacBinaryTranslationAvailable` indicates that while the conversion did succeed and that the source file was deleted, either:

- the `macCreator` and / or `macType` values were not recognized, and so the file suffix was defaulted to `.bin`
- the source file had no data fork

A return value of `kPGPError_NotMacBinary` indicates that the source file specification does not reference a MacOS MacBinary file. The source file is unaltered.

Syntax

```
PGPError PGPMacBinaryToLocal(
             PGPFileSpecRef inSpec,
             PGPFileSpecRef *outSpec,
             PGPUInt32 *macCreator,
             PGPUInt32 *macTypeCode );
```

Parameters

`inSpec`	the source file specification, which is assumed to reference a MacOS MacBinary file
`outSpec`	the receiving field for the file specification to the converted file
`macCreator`	the receiving field for the MacOS `OSType` of the creating application
`macType`	the receiving field for the MacOS `OSType` of the file type

Notes

The `macCreator` and `macType` arguments are optional. If specified as `NULL`, then the corresponding data item is not returned.

No assumption should be made regarding the name of the resultant file. The PGPsdk chooses the most appropriate extension for the encoded file type.

Preference Functions

PGPsdkLoadDefaultPrefs

Loads the preferences from the default preference file.

Syntax

```
PGPError PGPsdkLoadDefaultPrefs(
             PGPContextRef pgpContext );
```

Parameters

`pgpContext`	the target context

PGPsdkLoadPrefs

Loads the preferences from the specified preference file.

Syntax

```
PGPError PGPsdkLoadPrefs(
             PGPContextRef pgpContext,
             PGPFileSpecRef prefSpec );
```

Parameters

`pgpContext`	the target context
`prefSpec`	the file containing the stored preferences

PGPsdkSavePrefs

Saves any changed preference to its associated source file.

Syntax

```
PGPError PGPsdkSavePrefs(
            PGPContextRef pgpContext );
```

Parameters

pgpContext	the target context

Notes

The PGPContext "remembers" the source file from which each preference was loaded, and so the preference information is saved to that file.

PGPsdkPrefSetData

Sets the data associated with the specified preference to the specified (replacement) preference data.

Syntax

```
PGPError PGPsdkPrefSetData(
            PGPContextRef pgpContext,
            PGPsdkPrefSelector prefSelector,
            void const *prefBuf,
            PGPSize prefLength );
```

Parameters

pgpContext	the target context
prefSelector	the target preference
prefBuf	the associated (replacement) preference data
prefLength	the length of the associated (replacement) preference data

Notes

The caller must additionally call PGPsdkSavePrefs to make the change permanent.

PGPsdkPrefSetFileSpec

Establishes the specified file as the persistent store for the specified preference.

Syntax

```
PGPError PGPsdkPrefSetFileSpec(
             PGPContextRef pgpContext,
             PGPsdkPrefSelector prefSelector,
             PGPFileSpecRef fileSpec );
```

Parameters

`pgpContext`	the target context
`prefSelector`	the target preference
`fileSpec`	the (replacement) file specification

Notes

The caller must additionally call `PGPsdkSavePrefs` to make the change permanent.

PGPsdkPrefGetData

Retrieves the data associated with the specified preference into dynamically allocated memory.

Syntax

```
PGPError PGPsdkPrefGetData(
             PGPContextRef pgpContext,
             PGPsdkPrefSelector prefSelector,
             void **prefBuf,
             PGPSize *prefLength );
```

Parameters

`pgpContext`	the target context
`prefSelector`	the target preference
`prefBuf`	the receiving field for a pointer to the requested preference data
`prefLength`	the receiving field for the resultant length of the requested preference data

Notes

The caller is responsible for de-allocating the resultant preference data with `PGPFreeData.`

PGPsdkPrefGetFileSpec

Retrieves the file specification associated with the specified preference.

Syntax

```
PGPError PGPsdkPrefGetFileSpec(
             PGPContextRef pgpContext,
             PGPsdkPrefSelector prefSelector,
             PGPFileSpecRef *fileSpec );
```

Parameters

`pgpContext`	the target context
`prefSelector`	the target preference
`fileSpec`	the receiving field for the associated file specification

Notes

The caller is responsible for de-allocating the resultant file specification with `PGPFreeFileSpec`.

Date/Time Functions

PGPGetTime

Returns the current system time as a `PGPTime` format time value.

Syntax

```
PGPTime PGPGetTime( void );
```

Parameters

PGPGetPGPTimeFromStdTime

Returns the specified time as a `PGPTime` format time value.

Syntax

```
PGPTime PGPGetPGPTimeFromStdTime( time_t theTime );
```

Parameters

`theTime`	the time in Standard *C* Library time format

Notes

The data type `time_t` is that used by many of the Standard *C* Library time functions, for example `time()`.

PGPGetStdTimeFromPGPTime

Returns the specified `PGPTime` value as a `time_t` format time value.

Syntax

```
time_t PGPGetStdTimeFromPGPTime( PGPTime theTime );
```

Parameters

`theTime`	the time as a `PGPTime` data type

Notes

The data type `time_t` is that used by many of the Standard *C* Library time functions, for example `time()`.

PGPGetYMDFromPGPTime

Extracts the year, month, and day components from the specified `PGPTime` time value.

Syntax

```
void PGPGetYMDFromPGPTime(
             PGPTime theTime,
             PGPUInt16 *year,
             PGPUInt16 *month,
             PGPUInt16 *day );
```

Parameters

`theTime`	the time as a `PGPTime` data type
`year`	the receiving field for the year component
`month`	the receiving field for the month component
`day`	the receiving field for the day component

Notes

The `year`, `month`, and `day` arguments are optional. If specified as `NULL`, then the corresponding data item is not returned.

The `year` component includes the century.

The `month` and `day` components are one-based.

PGPTimeFromMacTime (*MacOS platforms only*)

Returns the specified MacOS format time value as a `PGPTime` format time value.

Syntax

```
PGPTime PGPTimeFromMacTime( PGPUInt32 theTime );
```

Parameters

`theTime`	the time as a MacOS format time value

PGPTimeToMacTime (*MacOS platforms only*)

Returns the specified `PGPTime` format `time` value as a MacOS format time value.

Syntax

```
PGPUInt32 PGPTimeToMacTime( PGPTime theTime );
```

Parameters

`theTime`	the time as a `PGPTime` format time value

Network Library Management Functions

PGPsdkNetworkLibInit

Function to initialize the PGPsdk network library. You should call this function early in your program, before calling any other network library function (the functions in `pgpKeyServer.h`, `pgpTLS.h`, and `pgpSockets.h`). This function can be called multiple times, but each successful call should be matched with a call to `PGPsdkNetworkLibCleanup()`.

Syntax

```
PGPError PGPsdkNetworkLibInit( void );
```

PGPsdkNetworkLibCleanup

Function to clean up the PGPsdk network library before exiting. This function can be called multiple times, and in fact should be called once for each successful call to `PGPsdkNetworkLibInit()`.

Syntax

```
PGPError PGPsdkNetworkLibCleanup( void );
```

Error Look-Up Functions

PGPGetErrorString

Looks-up the encoded error value, and places the corresponding error text formated as a *C* language string into the receiving buffer .

Syntax

```
PGPError PGPGetErrorString(
             PGPError theErrorCode,
             PGPsize availLength,
             char *theErrorText );
```

Parameters

`theErrorCode`	the encoded error value
`availLength`	the available length of the receiving buffer
`theErrorText`	the receiving buffer for the error text

Notes

The error text is truncated as required, and results in `kPGPError_BufferTooSmall` being returned.

`PGPGetErrorString()` is found in `pgpError.h`

Global Random Number Pool Management Functions

8

Introduction

Since the PGPsdk cryptographic functions require random numbers to operate correctly, the PGPsdk includes functions to manage a global pool of random numbers seeded from keystrokes and mouse movements. The SHA-1 hash function is used to distill entropy from incoming events and to spread it throughout the random pool.

The PGPsdk provides both cryptographically strong pseudo-random numbers as well as true random numbers based on external events. An internal fixed-size random pool holds random bits acquired from events passed in by the caller, and the PGPsdk estimates the entropy content (that is, the amount of true randomness) of the events, and tracks the total entropy available in the random pool at any time.

Random numbers are made available via an internal pseudo-random number generator (RNG) based on **ANSI X9.17**, and fed from the random pool. When there is sufficient entropy in the pool, the generator produces cryptographically strong true random numbers; when the entropy in the random pool is exhausted, the generator produces cryptographically strong pseudo-random numbers.

The ANSI X9.17 -compliant PGPsdk random number package includes the following functionality:

- acquiring randomness from environmental events passed in by the application
- filling buffers with random data as requested
- tracking the number of true random bits available

The random number functions support the following arguments and features to control their actions:

- random seeding from keystrokes and mouse movements
- a cryptographically strong pseudo-random number generator based on ANSI X9.17
- saving of the random pool state in persistent storage with reload on library initialization

- soft degrade from true environmental random bits to cryptographically strong pseudo-random bits

Header Files

```
pgpRandomPool.h
```

Random Number Pool Management Functions

PGPGlobalRandomPoolAddKeystroke

Augments the random number pool based upon the value of the captured keystroke. A non-zero return value indicates that the operation increased the entropy of the random number pool.

Syntax

```
PGPUInt32 PGPGlobalRandomPoolAddKeystroke(
             PGPInt32 keyCode );
```

Parameters

`keyCode`	the key code of the captured keystroke value

PGPGlobalRandomPoolAddMouse

This function is now deprecated. Developers should use `PGPGlobalRandomPoolMouseMoved()`instead.

PGPGlobalRandomPoolMouseMoved

Augments the random number pool based upon the timing between mouse-move events. A non-zero return value indicates that the operation increased the entropy of the random number pool.

Syntax

```
PGPUInt32    PGPGlobalRandomPoolMouseMoved(void);
```

Notes

Call this function repeatedly upon receiving mouse-moved events in your application event loop.

Entropy Estimation Functions

PGPGlobalRandomPoolGetSize

Returns the current size of the global random number pool in bytes.

Syntax

```
PGPUInt32 PGPGlobalRandomPoolGetSize( void );
```

PGPGlobalRandomPoolGetEntropy

Returns a measure of the current entropy of the global random number pool. This value is meaningful for the PGPsdk developer only when compared against the value returned by `PGPGlobalRandomPoolGetMinimumEntropy`.

Syntax

```
PGPUInt32 PGPGlobalRandomPoolGetEntropy( void );
```

PGPGlobalRandomPoolGetMinimumEntropy

Returns the minimum allowable entropy of the global random number pool that will support generation of random or cryptographically strong pseudo-random numbers for signing and / or encryption.

Syntax

```
PGPUInt32    PGPGlobalRandomPoolGetMinimumEntropy( void );
```

PGPGlobalRandomPoolHasMinimumEntropy

Returns TRUE if the current entropy of the global random number pool is sufficient to generate random or cryptographically strong pseudo-random numbers for signing and / or encryption.

Syntax

```
PGPBoolean  PGPGlobalRandomPoolHasMinimumEntropy( void );
```

PGPGetKeyEntropyNeeded

Returns the amount of entropy needed to generate a (sub-)key according to the specified options.

Syntax

```
PGPUInt32 PGPGetKeyEntropyNeeded(
             PGPContextRef pgpContext,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Entropy specific options include:

- `PGPOKeyGenParams` (required)
- `PGPOKeyGenFast`

Notes

If generating a DSS/Elgamal key, call this function twice - once for the DSS key and once for the Elgamal key - and sum the results.

`PGPGetKeyEntropyNeeded()` is found in `pgpKeys.h`.

User Interface Functions 9

Introduction

The PGPsdk user interface functions allow sophisticated PGPsdk developers access to the same dialog functionality employed by PGPtools. These dialogs may be customized through the use of UI-specific option functions.

Common features include:

- the dialogs will dismiss *only* upon satisfactory acceptance of the requested information (*except* `PGPCollectRandomDataDialog`, which auto-dismisses). For example, a passphrase dialog will remain open until a valid passphrase has been supplied, or the user clicks on the cancel button or the close button
- if the user cancels the dialog or otherwise closes the window before completing the dialog, then the dialog function will return `kPGPError_UserAbort`
- all passphrase dialogs *must* include a `PGPOUIOutputPassphrase` option, and the user is resopnsible for freeing the resultant passphrase with `PGPFreeData`

Header Files

`pgpUserInterface.h`

User Interface Management Functions

PGPsdkUILibInit

Initializes the PGPsdk user interface library. *This function must be called prior to using any of the other user interface functions.*

Syntax

```
PGPError PGPsdkUILibInit( void );
```

Notes

This function can be called multiple times but each successful call should be matched ba a call to `PGPsdkUILibCleanup()`.

PGPsdkCleanup

Releases any and all resources held by the PGPsdk user interface library.

Syntax

```
PGPError PGPsdkUILibCleanup( void );
```

Notes

This function should be called once for each successful call to `PGPsdkUILibInit()`, and can be called multiple times.

User Interface Dialog Functions

PGPRecipientDialog

Presents a generic dialog for selecting a set of recipient keys from a key set of all potential recipients.

Syntax

```
PGPError PGPRecipientDialog(
             PGPContextRef pgpContext,
             PGPKeySetRef allKeys,
             PGPBoolean alwaysDisplayDialog,
             PGPKeySetRef *recipientKeys,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`allKeys`	the key set containing all potential recipients
`alwaysDisplayDialog`	`TRUE` if the dialog should be displayed regardless of any `PGPOUIDefaultRecipients` option
`recipientKeys`	the receiving field for the resultant recipients key set
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`

- `PGPOUIEnforceAdditionalRecipientRequests`
- `PGPOKeyServerUpdateParams`
- `PGPOUIParentWindowHandle`
- `PGPOUIRecipientGroups`
- `PGPOUIWindowTitle`
- `PGPOUIDefaultRecipients`
- `PGPOUIRecipientGroups`
- `PGPOUIIgnoreMarginalValidity`
- `PGPOUIDisplayMarginalValidity`

Notes

This dialog may also behave in a non-visible / non-interactive mode to yield a default key set that meets specified validity requirements. To use the dialog in this manner, the caller must specify:

- `alwaysDisplayDialog` as `FALSE`
- a default key set with `PGPOUIDefaultRecipients` and / or `PGPOUIRecipientGroups`

  and the specified default key set must meet the following criteria:
- each key in the default key set must match exactly one key in the the key set containing all potential recipients
- each matched key is completely or marginally valid, depending upon the setting of `PGPOUIIgnoreMarginalValidity`

PGPPassphraseDialog

Presents a generic dialog for collecting a single passphrase.

Syntax

```
PGPError PGPPassphraseDialog(
             PGPContextRef  pgpContext,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIMinimumPassphraseLength`
- `PGPOUIMinimumPassphraseQuality`
- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

PGPConfirmationPassphraseDialog

Presents a dialog for collecting and verifying a passphrase.

Syntax

```
PGPError PGPConfirmationPassphraseDialog(
             PGPContextRef  pgpContext,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIMinimumPassphraseLength`
- `PGPOUIMinimumPassphraseQuality`
- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIShowPassphraseQuality`
- `PGPOUIWindowTitle`

PGPKeyPassphraseDialog

Presents a dialog for collecting and verifying the passphrase associated with a specific key.

Syntax

```
PGPError PGPKeyPassphraseDialog(
             PGPContextRef  pgpContext,
             PGPKeyRef key,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`key`	the target key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

PGPSigningPassphraseDialog

Presents a dialog for selecting a signing key and verifying its passphrase.

Syntax

```
PGPError PGPSigningPassphraseDialog(
             PGPContextRef pgpContext,
             PGPKeySetRef  allKeys,
             PGPKeyRef   *signingKey,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`allKeys`	the key set containing all potential signing

	keys
`recipientKeys`	the receiving field for the resultant signing keys key set
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDefaultKey`
- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIFindMatchingKey`
- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIVerifyPassphrase`
- `PGPOUIWindowTitle`

Notes

If the signing key set contains only split keys, then the function returns `kPGPError_KeyUnusableForSignature`.

PGPDecryptionPassphraseDialog

Syntax

```
PGPError PGPDecryptionPassphraseDialog(
             PGPContextRef pgpContext,
             PGPKeySetRef  recipientKeys,
             PGPUInt32 keyIDCount,
             const PGPKeyID keyIDList[],
             PGPKeyRef  *decryptionKey,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`recipientKeys`	the recipient key set
`keyIDCount`	the number of key IDs in the list
`keyIDList`	the list of keyIDs
`decryptionKey`	the receiving field for the resultant decryption

	key
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDefaultKey`
- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIFindMatchingKey`
- `PGPOUIKeyServerUpdateParams`
- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIVerifyPassphrase`
- `PGPOUIWindowTitle`

Notes

If the recipient key set contains only split keys, then the function returns `kPGPError_KeyUnusableForSignature`.

PGPConventionalEncryptionPassphraseDialog

Presents a dialog for selecting an encryption key and verifying its passphrase.

Syntax

```
PGPError PGPConventionalEncryptionPassphraseDialog(
            PGPContextRef  pgpContext,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`

- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

PGPConventionalDecryptionPassphraseDialog

Presents a dialog for specifying the passphrase associated with the key used to conventionally encrypt message.

Syntax

```
PGPError PGPConventionalDecryptionPassphraseDialog(
             PGPContextRef  pgpContext,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIOutputPassphrase` (required)
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

PGPOptionsDialog

Syntax

```
PGPError PGPOptionsDialog(
             PGPContextRef  pgpContext,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUICheckbox`
- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIParentWindowHandle`
- `PGPOUIPopupList`
- `PGPOUIWindowTitle`

PGPCollectRandomDataDialog

Presents a dialog that accumulates entropy bits from user mouse movements. Normally, this dialog appears as a response to an event of type `kPGPEvent_EntropyEvent`, or to a return of `FALSE` from `PGPGlobalRandomPoolHasMinimumEntropy`.

Syntax

```
PGPError PGPCollectRandomDataDialog(
             PGPContextRef pgpContext,
             PGPUInt32 neededEntropyBits,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`neededEntropyBits`	the number of entropy bits to be collected
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

- `PGPODialogPrompt`
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

Notes

This dialog auto-dismisses when enough entropy bits have been collected. A return value of `kPGPError_UserAbort` should be returned rarely, if ever, since simply moving the mouse to the cancel or close button is often sufficient to satisfy the specfied entropy requirement.

Attempts to collect less than approximately 500 entropy bits may result in such rapid auto-dismissal that the dialog appears to "flash" on the screen.

PGPSearchKeyServerDialog

Presents a dialog that specifies a set of keys to be transferred from one or more key servers. Upon return, all keys meeting the selection criteria are placed into the key set indicated by `foundKeys` (see `PGPQueryKeyServer`).

Syntax

```
PGPError PGPSearchKeyServerDialog(
            PGPContextRef pgpContext,
            PGPUInt32 keyServerCount,
            const PGPKeyServerSpec
               keyServerList[],
            PGPtlsContextRef tlsContext,
            PGPBoolean  searchAllKeyServers,
            PGPKeySetRef  *foundKeys,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`keyServerList`	the list of key servers to search
`keyServerCount`	the number of key servers in the list
`tlsContext`	the active TLS context
`searchAllKeyServers`	`TRUE` if all key servers should be searched; `FALSE` if the search should stop on the first match
`foundKeys`	the receiving field for the key set containing the resultant matching keys
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIKeyServerSearchAllServers`
- `PGPOUIKeyServerSearchFilter`
- `PGPOUIKeyServerSearchKey`
- `PGPOUIKeyServerSearchKeySet`
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

Notes

The `PGPOUIKeyServerUpdateParams` option is *not* valid for this function, since the option arguments essentially duplicate the function arguments.

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPSendToKeyServerDialog

Presents a dialog that specifies a set of keys to be transferred to a particular key server. Upon return, any keys that were not acceptable to the key server are placed into the key set indicated by `failedKeys` (see `PGPUploadToKeyServer`).

Syntax

```
PGPError PGPSendToKeyServerDialog(
             PGPContextRef pgpContext,
             const PGPKeyServerSpec *keyServer,
             PGPtlsContextRef tlsContext,
             PGPKeySetRef keysToSend,
             PGPKeySetRef  *failedKeys,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`keyServer`	the destination key server
`tlsContext`	the active TLS context
`keysToSend`	a key set containing the keys to send to the specified server
`failedKeys`	the receiving field for the key set containing those keys that were not accepted by the target

	key server
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Function specific options include:

- `PGPOUIDialogOptions`
- `PGPOUIDialogPrompt`
- `PGPOUIParentWindowHandle`
- `PGPOUIWindowTitle`

Notes

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

Misc. UI Functions

PGPEstimatePassphraseQuality

Returns a value in the range 0 (zero) to 100 which crudely estimates the "quality" of the specified passphrase, that is, its ability to resist known methods of attack. For example, the passphrase ***ABCD*** would yield a very low quality estimate while the passphrase ***Set course: star system NGC-13456-K*** would yield a very high quality estimate.

Syntax

```
PGPUInt32 PGPEstimatePassphraseQuality(
             const char *passphrase );
```

Parameters

`passphrase`	the target passphrase

Notes

This function provides "after the fact" determination of passphrase quality. The passphrase dialogs that solicit new passphrases accept options specifying minumum length and quality requirements (`PGPOUIMinimumPassphraseLength` and `PGPOUIMinimumPassphraseQuality`), as well as provide an option to display the passphrase quality as it is being entered (`PGPOUIShowPassphraseQuality`).

Key Server Functions 10

Introduction

The PGPsdk includes functions that support communication with HTTP and LDAP key servers, and allow developers to search for, add, disable, and delete keys on those servers.

Key server search operations support the same key filter mechanism described in Chapter 2, "Key Management Functions.", and yield a key set of the keys on the server that satisfy the filter criteria. LDAP servers support almost all of the available primitive key filters; HTTP servers support only a small number of the available primitive key filters (see Table 10-1 on page 244).

Key server add, disable, and delete operations accept a key set that specifies input, and yield a resultant key set that contains the keys that could not be added, disabled, or deleted.

A key server may have an associated user-defined event handler. The intent and functionality of this callback mechanism is similar to that of the event handler mechanism provided for key generation and encrypt/decrypt operations. If the callback function returns a value other than `kPGPError_NoErr`, then the associated key server operation is aborted.

A key server may also have an associated user-defined idle event handler. This function gains control periodically, and so allows the developer to look for a pending user cancel request, effect other processing as required, or perform whatever operations the developer wishes. This is particularly useful for operations that take a significant amount of time, such as search, add, disable, and delete operations. It is important to note that the intent and functionality of this callback mechanism is quite different from that of the event handler mechanism provided for key generation and encrypt/decrypt operations. No event is sent and no event-specific data is included – the callback function simply assumes control and executes until it returns. If the callback function returns a value other than `kPGPError_NoErr`, then the associated key server operation is aborted.

Header Files

`pgpKeyServer.h`

Constants and Data Structures

Table 10-1. Valid `PGPQueryKeyServer` Filters by Key Server Protocols

Filter Function	HTTP	LDAP
PGPIntersectFilters		•
PGPNegateFilter		•
PGPNewKeyCreationTimeFilter		•
PGPNewKeyDisabledFilter		•
PGPNewKeyEncryptAlgorithmFilter		•
PGPNewKeyEncryptKeySizeFilter		•
PGPNewKeyExpirationTimeFilter		•
PGPNewKeyFingerPrintFilter		
PGPNewKeyIDFilter	•	•
PGPNewKeyRevokedFilter		•
PGPNewKeySigAlgorithmFilter		•
PGPNewKeySigKeySizeFilter		
PGPNewSigKeyIDFilter		•
PGPNewSubKeyIDFilter		•
PGPNewUserIDEmailFilter	•	•
PGPNewUserIDNameFilter	•	•
PGPNewUserIDStringFilter	•	•
PGPUnionFilters		•

Events and Callbacks

A number of the key server functions allow the calling application to request callbacks to track the progress of the request. These functions generally require a perceptible amount of execution time, regardless of the size of their target key set.

An event handler serves two purposes – it provides notification to the calling application that an event has occurred, and provides a mechanism for the calling application to affect processing (in a pre-defined manner). Notification includes a pointer to a `PGPEvent` data type that, depending on the type of event, provides detailed information about the cause of the event. The calling application can then respond appropriately, which may or may not intervene and affect the course of further processing. If the calling application wishes to intervene, then it can abort the request by returning an error code (a value other than `kPGPError_NoErr`).

All event handlers are declared as

```
PGPError myEvents( PGPContextRef pgpContext,
              PGPEvent *event,
              PGPUserValue userValue );
```

The pgpContext argument is the reference to the context of the function posting the event. The event argument references a PGPEvent data type as follows:

```
struct PGPEvent_
{
        PGPVersion      version;
        struct PGPEvent_*nextEvent;
        PGPJobRef       job;
        PGPEventType    type;
        PGPEventData    data;
};
typedef struct PGPEvent_ PGPEvent;
```

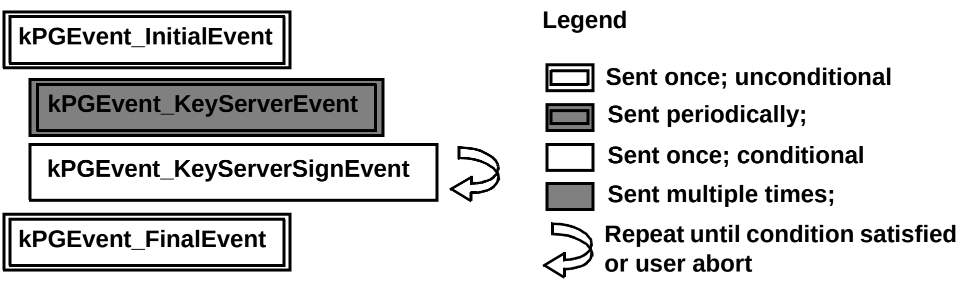
The version ad nextEvent members are currently reserved for internal use. The job member is not applicable to key server functions. The type member identifies the event being posted. The data member is a union of the event-specific data structures, which are described with their corresponding event.

The calling application can modify the processing context by invoking PGPAddJobOptions as:

```
PGPErrorPGPAddJobOptions( PGPJobRef job, ... );
```

The value of the job argument is that of the PGPEvent argument's job member. Additional PGPOptionListRef arguments are specified similarly to the way they are passed to PGPEncode and PGPDecode. However, only certain options can be set after each type of event, and these are listed for each event.

Figure 10-2. Key Server Request Processing Event Sequence



Key Server Request Events

kPGPEvent_InitialEvent

Sent before all other events. Implies initiation of the key server request.

Data

None

kPGPEvent_KeyServerEvent

Similar to `kPGPEvent_NullEvent,` this event reports the progress of the key server request, and allows the PGPsdk developer to determine its completion percentage.

The `state` member indicates the current point in the key server request processing from the caller's point of view.

The `soFar` and `total` members should be treated as relative, unscaled quantities – they are not necessarily byte or number-of-keys quantities.

Data

```
typedef struct PGPEventKeyServerData_
{
            PGPUInt32   state;
            PGPUInt32   soFar;
            PGPUInt32   total;
} PGPEventKeyServerData;
```

kPGPEvent_KeyServerSignEvent

Sent if a signing key is needed for authentication (posted by`PGPUploadToKeyserver`, `PGPDeleteFromKeyserver`, and `PGPDisableFromKeyserver`) to ensure that the requestor is authorized to effect the operation on the current qualifying key. The event handler should invoke `PGPAddJobOptions` specifying the `PGPOSignWithKey` and `PGPOClearSign` options, or return `kPGPError_UserAbort`. Note that `PGPOSignWithKey` further requires one the `PGPOPassphrase`, `PGPOPassphraseBuffer`, or `PGPOPasskeyBuffer` options

This event is sent repeatedly until a valid signing key is received, or until the event handler requests abort of the job. This allows the event handler to enforce a limit on the number of passphrase attempts.

The `state` member indicates the current point in the key server request processing from the caller's point of view, and assumes `kPGPKeyServerState`... values. It is not particularly useful in this context.

Data

```
typedef struct PGPEventKeyServerSignData_
{
```

```
            PGPUInt32   state;
} PGPEventKeyServerSignData;
```

kPGPEvent_FinalEvent

Sent after all other events. Implies completion of the key server request.

Data

None

Key Server Thread Storage

PGPKeyServerCreateThreadStorage

Allocates thread-local storage needed by the PGP key server routines and returns a reference to the existing storage for the current thread, if any.

Syntax

```
PGPError PGPKeyServerCreateThreadStorage(
            PGPKeyServerThreadStorageRef *prevStorage );
```

Parameters

`prevStorage`	the receiving field for a reference to existing storage in the current thread, if any.

Notes

The PGP key server utilities needs to keep “global” state for any threads actively using these socket calls. PGPsdk clients must call `PGPKeyServerCreateThreadStorage` to prepare a thread for using key server calls. When a client exits context, the state allocated by `PGPKeyServerCreateThreadStorage` must be disposed and the previous state restored using `PGPKeyServerDisposeThreadStorage`.

PGPKeyServerDisposeThreadStorage

Disposes thread-local storage allocated by `PGPKeyServerCreateThreadStorage` and restores the previous storage for the current thread, if any.

Syntax

```
PGPError PGPKeyServerDisposeThreadStorage(
            PGPKeyServerThreadStorageRef prevStorage );
```

Parameters

`prevStorage`	a reference to existing storage in the current thread, if any.

Key Server Functions

PGPKeyServerInit

Initializes the underlying communications layer that the PGPsdk requires for accessing a key server. This function effectively creates a communications session, and must be called prior to calling any other key server function.

Syntax

```
PGPError PGPKeyServerInit( void );
```

PGPNewKeyServerFromURL

This function is now deprecated. Developers should use `PGPNewKeyServer()` instead.

PGPNewKeyServerFromHostName

This function is now deprecated. Developers should use `PGPNewKeyServer()` instead.

PGPNewKeyServerFromHostAddress

This function is now deprecated. Developers should use `PGPNewKeyServer()` instead.

PGPNewKeyServer

Creates a new HTTP or LDAP communication context for the indicated host, depending on the specified options.

Syntax

```
PGPError PGPNewKeyServer(
             PGPContextRef pgpContext,
             PGPKeyServerClass class,
             PGPKeyServerRef *keyServerRef,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`pgpContext`	the target context
`class`	the class of the indicated key server (i.e. the key server product, such as PGP, NetTools CA, Verisign,

	Entrust, etc.)
`keyServerRef`	the receiving field for the resultant key server communication context
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

You must specify the server host by supplying one of the following three options, which are described in greater detail in the Option List Functions chapter of this document. Note that in all three cases, supplying a port number of `0` is interpreted as a request to use the indicated protocol's default port.

`PGPONetURL()`	Specifies the host by URL, expressed as a null-terminated C string in the following form: `[[protocol:]//]host.domain[:port]` Depending on the URL, the connection context will be either HTTP or LDAP. If the protocol: portion is omitted, then an HTTP context is assumed; if the `:port` portion is omitted, then an appropriate HTTP or LDAP port number is assumed.
`PGPONetHostName()`	Specifies the host by internet domain name, expressed as a null-terminated C string. Depending on the key server class, the connection context will be either HTTP or LDAP.
`PGPONetHostAddress()`	Specifies the host by internet domain address expressed as a 32-bit unsigned integer (i.e. 4 1-byte fields corresponding to the four parts of a ‘dotted quad’ such as `120.121.122.123`). Depending on the key server class, the connection context will be either HTTP or LDAP.

Other options include:

`PGPOKeyServerProtocol()`	If this option is omitted, then an HTTP context is assumed.
`PGPOKeyServerKeySpace()`	The area of the key server to access.This option is meaningful for LDAP key servers only, and indicates which keys may be acted upon by the following functions: `PGPQueryFromKeyserver()` `PGPDeleteFromKeyserver()`

`PGPDisableFromKeyserver()`

Providing a value of `kPGPKeyServerKeySpace_Normal` will restrict these functions to only those keys that satisfy the target key server's policy requirements; whereas a value of `kPGPKeyServerKeySpace_Pending` will restrict these functions to only those keys that haven't satisfied those policy requirements.

`PGPOKeyServerAccessType()`

Selects either normal or administrative access. This option is meaningful for LDAP key servers only, and is advisory only; that is, no initial authorization validation occurs. However, it must reflect `kPGPKeyServerAccess_Administrator` if the caller intends to later invoke any of the following functions:

```
PGPNewServerMonitor()
PGPUploadToKeyserver()
PGPDeleteFromKeyserver()
PGPDisableFromKeyserver()
```

PGPSetKeyServerEventHandler

Establish the specified function as the target key server's event handler.

Syntax

```
PGPError PGPSetKeyServerEventHandler(
            PGPKeyServerRef keyServer,
            PGPEventHandlerProcPtr callBack,
            PGPUserValue callBackArg );
```

Parameters

`keyServer`	the target key server
`callBack`	the desired non-idle event callback function or `NULL` to indicate no callbacks
`callBackArg`	the user-defined data, to be passed as an argument to any callback function

Notes

An event handler returning a value other than `kPGPError_NoError` will abort the current key server request.

For greatest flexibility, the PGPsdk developer should consider establishing `callBackArg` as a pointer to a user-defined data type, for example a *C* struct.

Specify `callBackArg` as `0` to indicate a dummy argument.

PGPGetKeyServerEventHandler

Retrieves the function pointer and callback argument of the target key server's non-idle event handler, if any. A resultant callback function value of `NULL` indicates that no callback function is defined; a resultant callback argument value of `0` indicates a dummy argument.

Syntax

```
PGPError PGPGetKeyServerEventHandler(
             PGPKeyServerRef keyServer,
             PGPEventHandlerProcPtr *callBack,
             PGPUserValue *callBackArg );
```

Parameters

`keyServer`	the target key server
`callBack`	the receiving field for the associated non-idle event handler function
`callBackArg`	the receiving field for the associated user-defined data, to be passed as an argument to any callback function

PGPSetKeyServerIdleEventHandler

Establish the specified function as the global idle event handler. For non-preemptive operating systems, this affords a mechanism for effecting yielding in threads. For pre-emptive operating systems, use of this function should be avoided, since it may interfere with the operating system's scheduling manager and actually impede performance.

Syntax

```
PGPError PGPSetKeyServerIdleEventHandler(
             PGPEventHandlerProcPtr callBack,
             PGPUserValue callBackArg );
```

Parameters

`callBack`	the desired idle event callback function or `NULL` to indicate no callbacks
`callBackArg`	the user-defined data, to be passed as an argument to any idle event callback function

Notes

The idle event handler you install will receive idle events for all currently active key servers.

For greatest flexibility, the PGPsdk developer should consider establishing `callBackArg` as a pointer to a user-defined data type, for example a *C* `struct`.

Specify `callBackArg` as `0` to indicate a dummy argument.

PGPGetKeyServerIdleEventHandler

Retrieves the function pointer and callback argument of the target key server's idle event handler, if any. A resultant callback function value of `NULL` indicates that no callback function is defined; a resultant callback argument value of `0` indicates a dummy argument.

Syntax

```
PGPError PGPGetKeyServerIdleEventHandler(
             PGPEventHandlerProcPtr *callBack,
             PGPUserValue *callBackArg );
```

Parameters

`callBack`	the receiving field for the associated callback function
`callBackArg`	the receiving field for the user-defined data, to be passed as an argument to any callback function

PGPGetKeyServerTLSSession

Retrieves the TLS session information for the specified key server (see `PGPNewTLSSession`).

Syntax

```
PGPError PGPGetKeyServerTLSSession(
             PGPKeyServerRef keyServer,
             PGPtlsSessionRef *tlsSession );
```

Parameters

`keyServer`	the target key server
`tlsSession`	the receiving field for the target key server's TLS session information

PGPGetKeyServerProtocol

Returns the protocol of the key server (HTTP, LDAP, etc.), as established when the key server reference was created. See `pgpKeyServer.h` for a list of supported protocols.

Syntax

```
PGPError PGPGetKeyServerProtocol(
             PGPKeyServerRef keyServer,
             PGPKeyServerProtocol *protocol );
```

Parameters

`keyServer`	the target key server
`protocol`	the receiving field for the target key server's protocol information

PGPGetKeyServerAccessType

Retrieves the access type for the specified key server. Specifically, this function provides a mechanism for determining if the key server connection was established with administrator (`kPGPKeyServerKeyAccess_Administrator`) access, which is required for certain requests.

Syntax

```
PGPError PGPGetKeyServerAccessType(
             PGPKeyServerRef keyServer,
             PGPKeyServerAccessType *accessType );
```

Parameters

`keyServer`	the target key server
`accessType`	the receiving field for the target key server's access type

PGPGetKeyServerKeySpace

Retrieves the key space for the specified key server. Specifically, this function provides a mechanism for determining if the key server connection was established to operate on keys that do meet policy requirements (`kPGPKeyServerKeySpace_Normal`) or that do *not* meet policy requirements (`kPGPKeyServerKeySpace_Pending`) .

Syntax

```
PGPError PGPGetKeyServerKeySpace(
             PGPKeyServerRef keyServer,
             PGPKeyServerKeySpace * keySpace );
```

Parameters

`keyServer`	the target key server
`keySpace`	the receiving field for the target key server's key space value

Notes

This function is meaningful for LDAP key servers only. HTTP key servers do not support the notion of "key space", and so this function is an effective no-op.

PGPGetKeyServerPort

Retrieves the port number of the specified key server's host connection.

Syntax

```
PGPError PGPGetKeyServerPort(
             PGPKeyServerRef keyServer,
             PGPInt16 * port );
```

Parameters

`keyServer`	the target key server
`port`	the receiving field for the port number of the target key server's host connection

PGPGetKeyServerHostName

Retrieves the host name of the specified key server's host.

Syntax

```
PGPError PGPGetKeyServerHostName(
             PGPKeyServerRef keyServer,
             char **hostName );
```

Parameters

`keyServer`	the target key server
`hostName`	the receiving field for the name of the target key server's host

Notes

The caller is responsible for de-allocating the resultant host name with `PGPFreeData`.

PGPGetKeyServerAddress

Retrieves the address of the specified key server's host connection.

Syntax

```
PGPError PGPGetKeyServerAddress(
             PGPKeyServerRef keyServer,
             PGPUInt32 *hostAddress );
```

Parameters

`keyServer`	the target key server
`hostAddress`	the receiving field for the address of the target key server's host

PGPGetKeyServerPath

Returns the file system path of the indicated key server's executable, reckoned from the machine's root. Note that this path is only available for servers created via URL; the returned path is the portion of the URL that follows the host and port specifications.

Syntax

```
PGPError GPGetKeyServerPath(
            PGPKeyServerRef keyServer,
            char **pathBuf );
```

Parameters

`keyServer`	the target key server
`pathBuf`	address of a pointer to an allocated buffer containing the path, which is expressed as a null-terminated C string.

Notes

Use `PGPFreeData()` to free the `pathBuf` when you're done with it.

PGPGetKeyServerContext

Returns the PGPContextRef that was used to create the indicated server reference.

Syntax

```
PGPContextRef GPGetKeyServerContext(
            PGPKeyServerRef keyServer, );
```

Parameters

`keyServer`	the target key server

PGPNewServerMonitor (*LDAP key servers only*)

Creates a new key server monitor that contains relevant data about and statistics for the specified LDAP key server. The resultant data and statistics are contained in a linked list of `PGPKeyServerMonitor` datatypes, which contain name/value pairs where a pair may have multiple values.

Depending upon the policies established for the target key server, this function may generate a `kPGPEvent_KeyServerSignEvent.` In this case, an associated event handler is required, or the function will fail with `kPGPError_ServerAuthorizationRequired` (see `PGPSetKeyServerEventHandler`).

Syntax

```
PGPError PGPNewServerMonitor(
            PGPKeyServerRef keyServer,
```

```
                PGPKeyServerMonitorRef *dataAndStats );
```

Parameters

`keyServer`	the target key server
`dataAndStats`	the receiving field for the resultant key server data and statistics

Notes

Calling this function for an HTTP key server will result in the return of `kPGPError_ServerOperationNotAllowed`.

The caller is responsible for de-allocating the resultant server monitor with `PGPFreeServerMonitor`.

PGPFreeServerMonitor (*LDAP key servers only*)

Decrements the reference count for the specified key server monitor, and de-allocatess the key server monitor if the reference count reaches zero.

Syntax

```
PGPError PGPFreeServerMonitor(
                PGPKeyServerMonitor * keyServerMonitor );
```

Parameters

`keyServer`	the target key server monitor

PGPFreeKeyServer

Decrements the reference count for the specified key server, and de-allocatess the key server if the reference count reaches zero.

Syntax

```
PGPError PGPFreeKeyServer(
                PGPKeyServerRef keyServer );
```

Parameters

`keyServer`	the target key server

PGPKeyServerOpen

Explicitly opens the specified key server. Key server request processing can be optimized by coding several key server requests within a `PGPKeyServerOpen` / `PGPKeyServerClose` "block", since this avoids implicit open / close operations for each request.

Syntax

```
PGPError PGPKeyServerOpen(
                PGPKeyServerRef keyServer,
```

```
PGPtlsSessionRef tlsSession );
```

Parameters

`keyServer`	the target key server
`tlsSession`	the active TLS context

Notes

This function is meaningful for LDAP key servers only. The HTTP protocol does not support the notion of "session", and so this function is an effective no-op.

A return value of `kPGPError_ServerSearchFailed` indicates that the target key server is not a certificate server, that is, it has no recognizable PGP key space.

The caller is responsible for explicitly closing the specified key server with `PGPKeyServerClose`.

PGPQueryKeyServer

Applies the specified key filter (constructed as detailed in Chapter 2, "Key Management Functions.") to the keys on the specified key server. This yields a resultant key set that contains all of the keys on the key server that meet the key filter criteria.

Syntax

```
PGPError PGPQueryKeyServer(
            PGPKeyServerRef keyServer,
            PGPFilterRef filter,
            PGPKeySetRef *resultSet );
```

Parameters

`keyServer`	the target key server
`filter`	the target key filter
`resultSet`	the receiving field for the resultant key set

Notes

`kPGPError_ServerOpenFailed` and `kPGPError_ServerSearchFailed` are returned for LDAP key servers only, and indicate that no `PGPKeyServerOpen` instance is currently in force.

The query may legitimately return an empty key set.

The caller is responsible for de-allocating the resultant key set (empty or not!) with `PGPFreeKeySet`.

PGPUploadToKeyServer

Transfers the specified keys to the specified key server. The key server connection must have been established with an access type of `kPGPKeyServerAccess_Administrator`.

Syntax

```
PGPError PGPUploadToKeyServer(
            PGPKeyServerRef keyServer,
            PGPKeySetRef keysToUpload,
            PGPKeySetRef *keysThatFailed );
```

Parameters

`keyServer`	the target key server
`keysToUpload`	the key set containing the keys to be transferred
`keysThatFailed`	the receiving field for the key set containing the keys that could not be successfully transferred

Notes

`kPGPError_ServerOpenFailed` and `kPGPError_ServerSearchFailed` are returned for LDAP key servers only if no `PGPKeyServerOpen` instance is currently in force.

Depending upon the policies established for the target key server, this function might generate a `kPGPEvent_KeyServerSignEvent` – potentially one for each key to be uploaded. In this case, a valid non-idle event handler is required, or the function will fail with `kPGPError_ServerAuthorizationRequired` (see `PGPSetKeyServerEventHandler`).

The returned error code is not always complete – multiple keys may have failed, each for a different reason. The choice of error code obeys the following hierarchy:

- key failed policy – usually indicates that the key was not signed by a recognized user.
- key already exists – the key data presented matches that already on the key server. This implies that the caller already has the most up-to-date version of the key
- key general failure
- other PGPsdk error code

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPDeleteFromKeyServer *(LDAP key servers only)*

Deletes the specified keys from the specified key server, which must be an LDAP key server. The key server connection must have been established with an access type of `kPGPKeyServerAccess_Administrator`.

Syntax

```
PGPError PGPDeleteFromKeyServer(
             PGPKeyServerRef keyServer,
             PGPKeySetRef keysToDelete,
             PGPKeySetRef *keysThatFailed );
```

Parameters

`keyServer`	the target key server
`keysToDelete`	the key set containing the keys to be deleted
`keysThatFailed`	the receiving field for the key set containing the keys that could not be successfully deleted

Notes

This function is *not* valid for HTTP key servers, and results in the return of `kPGPError_ServerOperationNotAllowed`.

`kPGPError_ServerOpenFailed` and `kPGPError_ServerSearchFailed` are returned for LDAPkey servers only if no `PGPKeyServerOpen` instance is currently in force.

Depending upon the policies established for the target key server, this function might generate a `kPGPEvent_KeyServerSignEvent` – potentially one for each key to be deleted. In this case, a valid non-idle event handler is required, or the function will fail with `kPGPError_ServerAuthorizationRequired` (see `PGPSetKeyServerEventHandler`).

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPDisableFromKeyServer *(LDAP key servers only)*

Disables the specified keys on the specified key server, which must be an LDAP key server. The key server connection must have been established with an access type of `kPGPKeyServerAccess_Administrator`.

Syntax

```
PGPError PGPDisableFromKeyServer(
             PGPKeyServerRef keyServer,
             PGPKeySetRef keysToDisable,
             PGPKeySetRef *keysThatFailed );
```

Parameters

`keyServer`	the target key server
`keysToDisable`	the key set containing the keys to be disabled
`keysThatFailed`	the receiving field for the key set containing the keys that could not be successfully disabled

Notes

This function is *not* valid for HTTP key servers, and results in the return of `kPGPError_ServerOperationNotAllowed`.

`kPGPError_ServerOpenFailed` and `kPGPError_ServerSearchFailed` are returned for LDAP key servers only if no `PGPKeyServerOpen` instance is currently in force.

Depending upon the policies established for the target key server, this function may generate a `kPGPEvent_KeyServerSignEvent` – potentially one for each key to be disabled. In this case, a valid non-idle event handler is required, or the function will fail with `kPGPError_ServerAuthorizationRequired` (see `PGPSetKeyServerEventHandler`).

The caller is responsible for de-allocating the resultant key set with `PGPFreeKeySet`.

PGPSendGroupsToServer (*LDAP key servers only*)

Uploads the specified key groups to the specified key server, which must be an LDAP key server. The key server connection must have been established with an access type of `kPGPKeyServerAccess_Administrator`.

Syntax

```
PGPError PGPSendGroupsToServer(
             PGPKeyServerRef keyServer,
             PGPGroupSetRef groupsToSend );
```

Parameters

`keyServer`	the target key server
`keysToDisable`	the key set containing the keys to be disabled
`groupsToSend`	the group set containg the key groups to uploadt

Notes

This function is *not* valid for HTTP key servers, and results in the return of `kPGPError_ServerOperationNotAllowed`.

Depending upon the policies established for the target key server, this function might generate a `kPGPEvent_KeyServerSignEvent` – potentially one for each key in each group to be uploaded. In this case, a valid non-idle event handler is required, or the function will fail with `kPGPError_ServerAuthorizationRequired`.

`kPGPError_ServerOpenFailed` and `kPGPError_ServerSearchFailed` are returned only if no `PGPKeyServerOpen` instance is currently in force.

PGPRetrieveGroupsFromServer *(LDAP key servers only)*

Retrieves all key groups from the specified key server, which must be an LDAP key server. The key server connection must have been established with an access type of `kPGPKeyServerAccess_Administrator`.

Syntax

```
PGPError PGPRetrieveGroupsFromServer(
             PGPKeyServerRef keyServer,
             PGPGroupSetRef *groups );
```

Parameters

`keyServer`	the target key server
`groups`	the receiving field for the resultant group set

Notes

This function is *not* valid for HTTP key servers, and results in the return of `kPGPError_ServerOperationNotAllowed`.

`kPGPError_ServerOpenFailed` and `kPGPError_ServerSearchFailed` are returned only if no `PGPKeyServerOpen` instance is currently in force.

The caller is responsible for de-allocating the resultant group set with `PGPFreeGroupSet`.

PGPSendCertificateRequest

Requests an X.509 certificate for a given key from the indicated CA server.

To retrieve the CA's response to your request, you should call `PGPRetrieveCertificate()` some time later. In rare cases a key server may respond quickly, but generating a certificate typically takes hours or days because in most organizations it requires a human Certification Authority to conduct due diligence research on the applicant.

Syntax

```
PGPError PGPSendCertificateRequest(
             PGPKeyServerRef keyServerRef,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`keyServer`	the target CA server
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final

	argument to terminate the argument list

Options

This function requires three entries to be present in the option list:

`PGPOKeyServerCAKey()`	selects a CA on the indicated server
`PGPOKeyServerRequestKey()`	provide the same key embedded in your cerificate request

You can use either of the following two options to furnish the formatted certificate request; but one of them is required:

`PGPOInputFile()`

`PGPOInputBuffer()`

Notes

Note that this function requires a properly formatted x509 certificate request. For guidance on creating a certificate request, please contact PGPsdk developer support.

PGPRetrieveCertificate

Retrieve X.509 certificate from a CA server, after its issuance by that CA. Typically the certificate will have been issued in response to an earlier request made by your program with `PGPSendCertificateRequest()`. The key to retrieve is specified using either an option based on its `PGPKeyRef`, or a search filter. Note that, unfortunately, the options and semantics required will differ slightly for each of the supported CA's.

Syntax

```
PGPError PGPRetrieveCertificate(
            PGPKeyServerRef keyServerRef,
            PGPOptionListRef firstOption,
            ...,
            PGPOLastOption() );
```

Parameters

`keyServer`	the target key server
`firstOption`	the initial option list instance
`...`	subsequent `option` list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

This function requires several entries to be present in the option list:

`PGPOKeyServerCAKey()`	selects a CA on the indicated server
`PGPOSignWithKey()`	provide the same key you searched

	with, and its passphrase

You can use either of the two following options to specify the key to retrieve, but one of them is required:

`PGPOKeyServerSearchKey()`	provide the same key you used in your certificate request; this key must be on the local keyring
`PGPOKeyServerSearchFilter()`	alternatively, you can search the server for the desired key

You can use any of the following three options to specify what to do with the retrieved certificate; but one of them is required:

`PGPOOutputFile()`

`PGPOOutputBuffer()`

`PGPOOutputAllocatedBuffer()`

Notes

Do not discard the output of this operation (i.e. do not use `PGPOOutputDiscard()` as an output option).

PGPRetrieveCertificateRevocationList

Retrieves any available X.509 certificate revocation lists (CRL) for the indicated key set from the indicated CA server.

Syntax

```
PGPError PGPRetrieveCertificateRevocationList(
             PGPKeyServerRef keyServerRef,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`keyServer`	the target key server
`firstOption`	the initial option list instance
`...`	subsequent option list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

This function requires several entries to be present in the option list:

`PGPOKeyServerCAKey()`	selects a CA on the indicated server
`PGPOKeySetRef()`	the set of keys for which to check for CRLs
`PGPOSignWithKey()`	provide the same key you searched

	with, and its passphrase

You can use either of the two following options, but one of them is required:

`PGPOKeyServerSearchKey()`	provide the same key you used in your request; this key must be on the local keyring
`PGPOKeyServerSearchFilter()`	alternatively, you can search the server for the desired key

You can use any of the following three options to specify what to do with the retrieved certificate; but one of them is required:

`PGPOOutputFile()`

`PGPOOutputBuffer()`

`PGPOOutputAllocatedBuffer()`

Notes

Do not discard the output of this operation (i.e. do not use `PGPOOutputDiscard()` as an output option).

PGPIncKeyServerRefCount

Increments the reference count of the specified key server. This provides a mechanism for manually incrementing the reference count should it be necessary.

Syntax

```
PGPError PGPIncKeyServerRefCount(
             PGPKeyServerRef keyServer );
```

Parameters

`keyServer`	the target key server

Notes

The PGPsdk automatically tracks the number of data items pointing to a particular resource. For example, a given key set may be referenced by any number of key lists and/or key iterators. This not only results in a level of context independence, but also ensures that a resource's memory is released only when its last reference is deleted. The PGPsdk also provides functions to support manual adjustment of a data item's reference count.

PGPGetLastKeyServerErrorString

Places the equivalent error text of the most recent error of the specified key server in the dynamically allocated string buffer.

This used to be just `PGPGetKeyServerErrorString` (and was consistent with `PGPGetErrorString`).

Syntax

```
PGPError PGPGetLastKeyServerErrorString(
            PGPKeyServerRef keyServer,
            char **theString );
```

Parameters

keyServer	the target key server
theString	the receiving field for a pointer to the associated error text

Notes

The caller is responsible for de-allocating the resultant error text with `PGPFreeData`.

If the most recent error has no associated error string, then the function returns `kPGPError_NoError`, and `theString` will be `NULL`.

PGPCancelKeyServerCall

Syntax

```
PGPError PGPCancelKeyServerCall(
            PGPKeyServerRef keyServer );
```

Parameters

keyServer	the target key server

Notes

Once return has been made from a canceled call, the target key server must be closed with `PGPKeyServerClose`.

PGPKeyServerClose

Explicitly closes the specified key server (see `PGPKeyServerOpen`).

Syntax

```
PGPError PGPKeyServerClose(
            PGPKeyServerRef keyServer );
```

Parameters

keyServer	the target key server

Notes

This function is meaningful for LDAP key servers only. The HTTP protocol does not support the notion of “session”, and so this function is an effective no-op.

PGPKeyServerCleanup

Terminates the underlying communications layer that the PGPsdk requires for accessing a key server (see `PGPKeyServerInit`). This function effectively destroys a communications session, and so `PGPKeyServerInit` must be called to initiate a new session prior to calling any other key server function.

Syntax

```
PGPError PGPKeyServerCleanup( void );
```

TLS (Transport Layer Security) Functions 11

Introduction

The PGPsdk TLS (Transport Layer Security) functions allow sophisticated PGPsdk developers access to the underlying functions that form the basis for secure communication between the client application and the remote key server. These include:

- create, manage, and free TLS contexts and sessions
- attach a socket to a TLS session

Header Files

`pgpTLS.h`

TLS Context Management Functions

PGPNewTLSContext

Creates a new TLS context, which has caching enabled.

Syntax

```
PGPError PGPNewTLSContext(
            PGPContextRef pgpContext,
            PGPtlsContextRef *tlsContext );
```

Parameters

`pgpContext`	the target context
`tlsContext`	the receiving field for the resultant TLS context

Notes

The caller is responsible for deallocating the resultant TLS context with `PGPFreeTLSContext`.

Use `PGPtlsSetCache` to override the caching default.

PGPFreeTLSContext

Frees the specified TLS context.

Syntax

```
PGPError PGPFreeTLSContext(
             PGPtlsContextRef tlsContext );
```

Parameters

`tlsContext`	the target TLS context

PGPtlsSetCache

Activates or deactivates the session key cache for sessions created using the specified TLS context, depending upon the value specified for `useCache`.

Syntax

```
PGPError PGPtlsSetCache(
             PGPtlsContextRef pgpContext,
             PGPBoolean useCache );
```

Parameters

`pgpContext`	the target context
`useCache`	set to TRUE to enable use of the cache; set to FALSE to disable use of the cache

Notes

Cache usage defaults to TRUE upon context creation (see `PGPNewTLSContext`).

PGPtlsClearCache

Resets the session key cache for all sessions created using the specified TLS context.

Syntax

```
PGPError PGPtlsClearCache(
             PGPtlsContextRef pgptlsContext );
```

Parameters

`pgptlsContext`	the target TLS context

Notes

Context creation uses any existing cache (see `PGPNewTLSContext`).

PGPNewTLSSession

Creates a new TLS session.

Syntax

```
PGPError PGPNewTLSSession(
            PGPtlsContextRef tlsContext,
            PGPtlsSessionRef *tlsSession );
```

Parameters

`pgptlsContext`	the target TLS context
`tlsSession`	the receiving field for the resultant TLS session

Notes

The caller is responsible for deallocating the resultant TLS session with `PGPFreeTLSSession`.

The session protocol options default to:

- `kPGPtlsFlags_ClientSide`
- `!( kPGPtlsFlags_RequestClientCert )`

(see `PGPtlsSetProtocolOptions`).

PGPCopyTLSSession

Creates an exact copy of the source TLS session, including its current state.

Syntax

```
PGPError PGPCopyTLSSession(
            PGPtlsSessionRef tlsOrig,
            PGPtlsSessionRef *tlsCopy );
```

Parameters

`tlsOrig`	the source TLS session
`tlsCopy`	the receiving field for the copy of the TLS session

Notes

The caller is responsible for deallocating the resultant TLS session copy with `PGPFreeTLSSession`.

PGPtlsHandshake

Initiates a TLS session by performing all negotiation involved with establishing the actual TLS connection. No data may be sent or received by the session until this function returns `kPGPError_NoError` or `PGPtlsSession` reflects a session state of `kPGPtls_ReadyState`.

Syntax

```
PGPError PGPtlsHandshake(
             PGPtlsSessionRef tlsSession );
```

Parameters

`tlsSession`	the target TLS session

PGPtlsClose

Terminates a TLS session by performing all clean-up involved with tearing down the actual TLS connection. No data may be sent or received by the session after this call.

If `noSessionKeyCache` is specified as `TRUE`, then the session keys are not added to the cache (if any)

Syntax

```
PGPError PGPtlsClose(
             PGPtlsSessionRef tlsSession,
             PGPBoolean noSessionKeyCache );
```

Parameters

`tlsSession`	the target TLS session
`noSessionKeyCache`	indicates whether or not the session keys should be added to the cache (if any)

Notes

If the specified session is not terminated by this function or if `noSessionKeyCache` is specified as `TRUE`, then it cannot be restarted from the session cache.

If the client application determines that the connection has experienced errors, for example, the remote key is invalid, then this function should be called with `noSessionKeyCache` specified as `TRUE`.

PGPFreeTLSSession

Deallocates the specified TLS session.

Syntax

```
PGPError PGPFreeTLSSession(
             PGPtlsSessionRef tlsSession );
```

Parameters

`tlsSession`	the target TLS session

PGPtlsSetRemoteUniqueID

Sets the remote ID for the specified TLS session.

Syntax

```
PGPError PGPtlsSetRemoteUniqueID(
             PGPtlsSessionRef tlsSession,
             PGPUInt32 remoteID );
```

Parameters

`tlsSession`	the target TLS session
`remoteID`	the desired remote ID, which is nominally an IP address

Notes

This function *must* be called *prior* to `PGPtlsHandshake`.

PGPtlsSetProtocolOptions

Sets the protocol options for the specified TLS session.

Syntax

```
PGPError PGPtlsSetProtocolOptions(
             PGPtlsSessionRef tlsSession,
             PGPtlsFlags optionFlags );
```

Parameters

`tlsSession`	the target TLS session
`optionFlags`	the desired protocol option flags

Notes

This function *must* be called *prior* to `PGPtlsHandshake`.

PGPtlsSetDHPrime

Sets the Diffie-Hellman prime to one of the specified size (in bits). The requested primes are drawn from a set of hard-coded primes. New primes can be added in a fully compatible fashion since the server sends the prime to the client, but this version of the API does not support passing in a desired prime.

Syntax

```
PGPError PGPtlsSetDHPrime(
             PGPtlsSessionRef tlsSession,
             PGPtlsPrime primeSize );
```

Parameters

`tlsSession`	the target TLS session
`primeSize`	the desired Diffie-Hellman prime size (in bits)

Notes

This function *must* be called *prior* to `PGPtlsHandshake`.

The default prime if this function is not called is `kPGPtls_DHPrime1536`.

PGPtlsSetPreferredCipherSuite

Indicates which TLS cipher suite the client prefers to use for the session.

Syntax

```
PGPContextRef PGPtlsSetPreferredCipherSuite(
             PGPtlsSessionRef tlsSession,
             PGPtlsCipherSuiteNum cipher );
```

Parameters

`tlsSession`	the target TLS session
`cipher`	the desired cipher suite

Notes

This function *must* be called *prior* to `PGPtlsHandshake`.

This function indicates a preference only. Call `PGPtlsGetNegotiatedCipherSuite` once the session has been established to determine the actual cipher suite being used.

PGPtlsGetNegotiatedCipherSuite

Returns the identity of the TLS cipher suite that will be used use for the session.

Syntax

```
PGPContextRef PGPtlsGetNegotiatedCipherSuite(
             PGPtlsSessionRef tlsSession,
             PGPtlsCipherSuiteNum *cipher );
```

Parameters

`tlsSession`	the target TLS session
`cipher`	the desired cipher suite, which assumes `kPGPtls_TLS_...`values

Notes

This function *must* be called *subsequent* to `PGPtlsHandshake`.

PGPtlsSetLocalPrivateKey

Sets the local private authenticating key. The passphrase and key are retained in memory. By default, no key is specified and a client side session will return no key in the client key exchange message to the server. It is an error not to specify a key on a server side TLS session.

If you wish to validate your TLS connection with an X.509 certificate, the `X509Cert` parameter must refer to a valid X.509 certificate and the `CertChain` parameter must refer to a set of keys containing all keys in the certificate chain for the indicated X.509 certificate, going all the way up to the root Certification Authority. The `CertChain` keys must remain valid for the duration of the TLS connection.

To forego X.509 validation, pass in `kPGPInvalidSigRef` for the `X509Cert` parameter and `kInvalidPGPKeySetRef` for the `CertChain` parameter.

Syntax

```
PGPError PGPtlsSetLocalPrivateKey(
             PGPtlsSessionRef tlsSession,
             PGPKeyRef localPrivateKey,
             PGPSigRef X509Cert,
             PGPKeySetRef CertChain,
             PGPOptionListRef firstOption,
             ...,
             PGPOLastOption() );
```

Parameters

`tlsSession`	the target TLS session
`localPrivateKey`	the desired local private key
`X509Cert`	a valid X.509 certificate, or `kPGPInvalidSigRef`
`CertChain`	the set of keys in the complete certificate chain for the indicated X.509 certificate, or `kInvalidPGPKeySetRef` if no X.509 certificate was provided.
`firstOption`	the initial option list instance
`...`	subsequent `option` list instances
`PGPOLastOption()`	must always appear as the final argument to terminate the argument list

Options

Local private authenticating key specific options include:

- `PGPOPasskeyBuffer`
- `PGPOPassphrase`
- `PGPOPassphraseBuffer`

Notes

The PGPsdk internally treats and represents X.509 certificates as signatures on keys.

It is the developer's responsibility to obtain the X.509 certificate chain keys, and to form them into a key set.

PGPtlsGetRemoteAuthenticatedKey

Obtains the authenticated remote key after a performing successful handshake with `PGPtlsHandshake()`.

Syntax

```
PGPError PGPtlsGetRemoteAuthenticatedKey(
             PGPtlsSessionRef tlsSession,
             PGPKeyRef *remoteKey,
             PGPKeySetRef *X509CertChainKeys );
```

Parameters

`tlsSession`	the target TLS session
`remoteKey`	the receiving field for the authenticated remote key
`X509CertChainKeys`	the receiving field for X.509 certificate chain keys

Notes

The key returned must already have been approved through the callback mechanism. The PGPEvent mechanism is used to request approval from the client of the remote key received during the TLS handshake. The callback should be set through the standard PGPSockets callback mechanism. The event `kPGPEvent_TLSRemoteKeyApprovalEvent` will be used in this case. In some cases, the `kPGPEvent_TLSRemoteKeyApprovalEvent` may only pass a Key ID to the caller, and it will be up to the caller to resolve the Key ID into a key and pass the `PGPKeyRef` back to TLS.

For an X.509-validated TLS connection, the `X509CertChainKeys` parameter will be set to the complete set of keys in the certificate chain for the X.509 certificate, as provided via `PGPtlsSetLocalPrivateKey()`.

This function *must* be called *subsequent* to `PGPtlsHandshake`.

PGPtlsGetState

Returns the current state of the specified TLS session.

Syntax

```
PGPContextRef PGPtlsGetState(
             PGPtlsSessionRef tlsSession,
             PGPtlsProtocolState *sessionState );
```

Parameters

`tlsSession`	the target TLS session
`sessionState`	the receiving field for the session state value

Notes

PGPtlsGetAlert

Obtains the alert code of the fatal alert that caused the TLS session to abort and go into the `kPGPtls_FatalErrorState`.

Syntax

```
PGPError PGPtlsGetAlert(
             PGPtlsSessionRef tlsSession,
             PGPtlsAlert *alert );
```

Parameters

`tlsSession`	the target TLS session
`alert`	the receiving field for the alert code

Notes

This function should *not* be called unless `PGPtlsGetState` indicates `kPGPtls_FatalErrorState`.

PGPtlsSetSendCallback

Sets the send callback function to that specified.

Syntax

```
PGPError PGPtlsSetSendCallback(
            PGPtlsSessionRef tlsSession,
            PGPtlsSendProcPtr tlsSendProc,
            void *userData );
```

Parameters

`tlsSession`	the target TLS session
`tlsSendProc`	the desired send callback function
`userData`	user data needed by the callback function

PGPtlsSend

Sends data over the underlying `PGPsockets` connection.

Syntax

```
PGPError PGPtlsSend(
            PGPtlsSessionRef tlsSession,
            const void *outBuffer,
            PGPSize bufferLength );
```

Parameters

`tlsSession`	the target TLS session
`outBuffer`	the data to be sent
`bufferLength`	the size (in bytes) of the data to be sent

Notes

It is an error to call this function without having set a Write function pointer. Most applications will never need to use this function as the function pointers are automatically configured by `PGPsockets`, and this function is automatically called by the `PGPsockets` implementations of `PGPWrite` whenever a `PGPtlsSessionRef` has been set for a given socket.

PGPtlsSetReceiveCallback

Sets the receive callback function to that specified.

Syntax

```
PGPError PGPtlsSetReceiveCallback(
            PGPtlsSessionRef tlsSession,
            PGPtlsReceiveProcPtr tlsReceiveProc,
            void *userData );
```

Parameters

`tlsSession`	the target TLS session
`tlsReceiveProc`	the desired receive callback function
`userData`	user data needed by the callback function, if any

PGPtlsReceive

Retrieves data over the underlying PGPSockets connection.

Syntax

```
PGPError PGPtlsReceive(
             PGPtlsSessionRef tlsSession,
             void *inBuffer,
             PGPSize *bufferLength );
```

Parameters

`tlsSession`	the target TLS session
`inBuffer`	the receiving field for the incoming data
`bufferLength`	the size (in bytes) of the receiving buffer

Notes

It is an error to call this function without having set a Read function pointer. Most applications will never need to use this functions as the function pointers are automatically configured by PGPsockets, and this function is automatically called by the PGPsockets implementations of PGPRead whenever a `PGPtlsSessionRef` has been set for a given socket.

Socket Functions 12

Introduction

The PGPsdk socket functions allow sophisticated PGPsdk developers further access to the functions that form the basis for secure communication between PGP client and server applications. Based upon Berkeley sockets and WINSOCK Version 1.1 (although not WINSOCK compliant), the PGP socket layer provides a simple, platform independent abstraction (particularly for MacOS). However, the true motivation behind the PGP socket layer lies in employing it as an *encrypting* socket layer by associating it with an existing TLS session (see `PGPSocketsEstablishTLSSession`).

The PGP socket layer supports both stream and datagram sockets. Stream sockets provide for bi-directional, reliable, sequenced, and unduplicated data flow with no concept of record boundaries. Datagram sockets provide for bi-directional data flow with enforcement of record boundaries, but do not guarantee the data to be reliable, sequenced, or unduplicated.

Specific functional support includes:

- socket creation
- socket listen, bind and connect
- socket management
- data send / send to
- data receive / receive from
- socket deletion

Many of the PGP socket layer functions do not return `PGPError`. Rather, in keeping with the Berkeley sockets model, a return value of `kPGPSockets_Error` indicates that the operation failed. In this case, the caller must obtain the actual error code with `PGPGetLastSocketError`.

In keeping with the WINSOCK model, the PGP socket layer currently supports only the Internet domain (`kPGPAddressFamilyInternet`).

Header Files

pgpSockets.h

Constants and Data Structures

Table 12-1. WINSOCK Error Mappings

PGPsdk Constant	WINSOCK Constant
kPGPError_BadParams	WSAEFAULT
	WSAEDESTADDRREQ
	WSAENOPROTOOPT
	WSAESOCKTNOSUPPORT
	WSAVERNOTSUPPORTED
kPGPError_OutOfMemory	WSAENOBUFS
kPGPError_SocketsAddressFamilyNotSupported	WSAEAFNOSUPPORT
kPGPError_SocketsAddressInUse	WSAEADDRINUSE
kPGPError_SocketsAddressNotAvailable	WSAEADDRNOTAVAIL
kPGPError_SocketsAlreadyConnected	WSAEISCONN
kPGPError_SocketsBufferOverflow	WSAEMSGSIZE
kPGPError_SocketsDomainServerError	WSATRY_AGAIN
	WSANO_RECOVERY
	WSANO_DATA
kPGPError_SocketsHostNotFound	WSAHOST_NOT_FOUND
kPGPError_SocketsInProgress	WSAEINPROGRESS
kPGPError_SocketsNetworkDown	WSAENETDOWN
	WSAENETUNREACH
	WSASYSNOTREADY
kPGPError_SocketsNotASocket	WSAENOTSOCK
kPGPError_SocketsNotBound	WSAEINVAL
kPGPError_SocketsNotConnected	WSAECONNREFUSED
	WSAECONNABORTED
	WSAECONNRESET
	WSAENETRESET
kPGPError_SocketsNotInitialized	WSANOTINITIALISED
kPGPError_SocketsOperationNotSupported	WSAEOPNOTSUPP
kPGPError_SocketsProtocolNotSupported	WSAEPROTONOSUPPORT
	WSAEPROTOTYPE
kPGPError_SocketsTimedOut	WSAETIMEDOUT

Table 12-2. UNIX Socket Error Mapping

PGPsdk Constant	UNIX Constant
kPGPError_BadParams	EFAULT
	EDESTADDRREQ
	ENOPROTOOPT
	ESOCKTNOSUPPORT
kPGPError_OutOfMemory	ENOBUFS
kPGPError_SocketsAddressFamilyNotSupported	EAFNOSUPPORT
kPGPError_SocketsAddressInUse	EADDRINUSE
kPGPError_SocketsAddressNotAvailable	EADDRNOTAVAIL
kPGPError_SocketsAlreadyConnected	EISCONN
kPGPError_SocketsBufferOverflow	EMSGSIZE
kPGPError_SocketsDomainServerError	TRY_AGAIN
	NO_RECOVERY
	NO_DATA
kPGPError_SocketsHostNotFound	HOST_NOT_FOUND
kPGPError_SocketsInProgress	EINPROGRESS
kPGPError_SocketsNetworkDown	ENETDOWN
	ENETUNREACH
kPGPError_SocketsNotASocket	ENOTSOCK
kPGPError_SocketsNotBound	EINVAL
kPGPError_SocketsNotConnected	ECONNREFUSED
	ECONNABORTED
	ECONNRESET
	ENETRESET
kPGPError_SocketsOperationNotSupported	EOPNOTSUPP
kPGPError_SocketsProtocolNotSupported	EPROTONOSUPPORT
	EPROTOTYPE
kPGPError_SocketsTimedOut	ETIMEDOUT

Initialization and Termination Functions

PGPSocketsInit

Initializes the underlying sockets layer upon which the PGPsdk sockets layer depends. This must be called prior to calling any other PGPsdk sockets function. This function is reference counted and must be matched by an equal number of calls to `PGPSocketsCleanup`.

Syntax

```
PGPError PGPSocketsInit( void );
```

PGPSocketsCleanup

Terminates the underlying sockets layer upon which the PGPsdk sockets layer depends (see `PGPSocketsInit`). This precludes any further calls to PGPsdk sockets layer functions other than `PGPSocketsInit`.

Syntax

```
void PGPSocketsCleanup( void );
```

Socket Thread Storage

PGPSocketsCreateThreadStorage

Allocates thread-local storage needed by the PGP socket layer and returns a reference to the existing storage for the current thread, if any.

Syntax

```
PGPError PGPSocketsCreateThreadStorage(
             PGPSocketsThreadStorageRef *prevStorage );
```

Parameters

`prevStorage`	the receiving field for a reference to existing storage in the current thread, if any.

Notes

The PGP socket layer needs to keep “global” state for any threads actively using these socket calls. PGPsdk clients must call `PGPSocketsCreateThreadStorage` to prepare a thread for using the PGP socket layer. When a client exits context, the state allocated by `PGPSocketsCreateThreadStorage` must be disposed and the previous state restored using `PGPSocketsDisposeThreadStorage`.

PGPSocketsDisposeThreadStorage

Disposes thread-local storage allocated by `PGPSocketsCreateThreadStorage` and restores the previous storage for the current thread, if any.

Syntax

```
PGPError PGPSocketsDisposeThreadStorage(
             PGPSocketsThreadStorageRef prevStorage );
```

Parameters

`prevStorage`	a reference to existing storage in the current thread, if any.

Socket Creation and Destruction Functions

PGPOpenSocket

Creates a socket of the specified address family, type, and protocol. If the returned socket reference is `kInvalidPGPSocketRef`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

Syntax

```
PGPSocketRef PGPOpenSocket(
             PGPInt32 addressFamily,
             PGPInt32 socketType,
             PGPInt32 socketProtocol );
```

Parameters

`addressFamily`	the desired address family
`socketType`	the desired socket type
`socketProtocol`	the desired socket protocol

Notes

`PGPSocketsInit` must have been called prior to invoking this function.

If `addressFamily` is specified as `kPGPAddressFamilyUnspecified`, then `socketProtocol` may not be specified as `kPGPProtocolFamilyUnspecified`.

PGPSetSocketsIdleEventHandler

Sets the idle event handler for the currently selected sockets to that specified (see `PGPSelect`), which will receive periodic idle events during network calls. If the idle event handler returns other than `PGPError_NoErr`, the blocking socket will be automatically closed.

Syntax

```
PGPError PGPSetSocketsIdleEventHandler(
             PGPEventHandlerProcPtr callback,
             PGPUserValue callBackArg );
```

Parameters

`callBack`	the desired idle event handler function
`callBackArg`	user-defined data, to be passed to the idle event handler

Notes

An idle event handler is associated with one and only one thread.

Normally, the idle event handler is used only in non-preemptive multi-tasking operating systems, so that threads may periodically yield control. In pre-emptive multi-tasking systems, use of an idle event handler may adversely may adversely impact the existing scheduling algorithm(s).

PGPGetSocketsIdleEventHandler

Obtains the receive callback function currently defined for the currently selected sockets to that specified (see PGPSelect).

Syntax

```
PGPError PGPGetSocketsIdleEventHandler(
             PGPEventHandlerProcPtr *callback,
             PGPUserValue *callBackArg );
```

Parameters

callBack	the receiving field for the idle event handler function
callBackArg	user-defined data, to be passed to the idle event handler

PGPCloseSocket

Closes the specified socket. If the return value is kPGPSockets_Error, then the caller should obtain the actual error code via PGPGetLastSocketError.

Syntax

```
PGPInt32 PGPCloseSocket( PGPSocketRef socketRef );
```

Parameters

socketRef	the target socket

Notes

PGPSocketsInit must have been called prior to invoking this function.

A resultant error of kPGPError_SocketsNotASocket may indicate that the socket has been previously closed.

Endpoint Binding Functions

PGPBindSocket

Binds the specified socket, which must be unbound and unconnected, to the specified address. This establishes a local name association for the socket, which in turn establishes a local association with the socket's address.

Syntax

```
PGPInt32 PGPBindSocket(
             PGPSocketRef socketRef,
             const PGPSocketAddress *address,
             PGPInt32 addressLength );
```

Parameters

`socketRef`	the target socket
`address`	the bind-to address
`addressLength`	the length of the bind-to address, which is normally `sizeof( PGPSocketAddress )`

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPConnect

Connects the specified socket, which must be unconnected, to the specified address, which is assumed to be on a foreign host. Upon successful return, the socket is ready to effect send / receive operations.

If the target socket is unbound, the a system-generated name is assigned to the socket, and the socket is bound to that name.

Syntax

```
PGPInt32 PGPConnect(
            PGPSocketRef socketRef,
            const PGPSocketAddress *address,
            PGPInt32 addressLength);
```

Parameters

`socketRef`	the target socket
`address`	the connect-to address
`addressLength`	the length of the connect-to address, which is normally `sizeof( PGPSocketAddress )`

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

Server Functions

PGPListen

Creates a pending connections queue for the specified socket, which must be bound, but must not be connected.

Syntax

```
PGPInt32 PGPListen(
            PGPSocketRef socketRef,
            PGPInt32 maxBacklog );
```

Parameters

`socketRef`	the target socket
`maxBackLog`	the maximum length to which the pending connections queue may grow

Notes

`PGPSocketsInit` must have been called prior to invoking this function.

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPAccept

Creates a new socket having the the same characteristics as the specified template socket, and associates it with the first connection on the pending connection queue of the specified template socket. The template socket remains open.

If address in non-NULL and addressLength is non-zero, then they receive the address of the connecting entity.

Syntax

```
PGPSocketRef PGPAccept(
            PGPSocketRef socketRef,
            PGPSocketAddress *address,
            PGPInt32 *addressLength );
```

Parameters

`socketRef`	the template socket
`address`	the receive-from address (optional)
`addressLength`	the length of the receive-from address (optional)

Notes

`PGPSocketsInit` must have been called prior to invoking this function.

If the return value is `kInvalidPGPSocketsRef`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

The resultant new socket may *not* be presented subsequently to `PGPAccept` as a template socket.

PGPSelect

Determines the status of one or more sockets, and returns the number of sockets that meet the criteria, and which represents the total number of descriptors contained in the specified `PGPSocketSet` arguments after they have been updated. If the returned count is `kPGPSocket_Errror`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

Syntax

```
PGPInt32  PGPSelect(
             PGPInt32 numSetCount,
             PGPSocketSet *readSet,
             PGPSocketSet *writeSet,
             PGPSocketSet *errorSet,
             const PGPSocketsTimeValue *timeout );
```

Parameters

`numSetCount`	
`readSet`	a pointer to a set of sockets to be checked for readability, or `NULL` if no sockets are to be checked for readability
`writeSet`	a pointer to a set of sockets to be checked for writabilitya pointer to a set of sockets to be checked for readability, or `NULL` if no sockets are to be checked for readability
`errorSet`	a pointer to a set of sockets to be checked for the presence of out-of-band data or outstanding error conditionsa pointer to a set of sockets to be checked for readability, or `NULL` if no sockets are to be checked for readability
`timeout`	the desired timeout interval. A specification of `NULL` denotes a blocking operation; a `PGPSocketsTimeValue` of 0 (zero) denotes a non-blocking operation with immediate return (polling).

Notes

`PGPSocketsInit` must have been called prior to invoking this function.

`numSetCount` is used by UNIX platforms only; it is not used by Windows and MacOS platforms.

Out-of-band data is accessed via `errorSet`, since the PGPsdk socket layer does not support the OOBINLINE option.

A *readable* socket is one which:

- is listening
- has data queued
- is a stream socket that has been closed, and so will return zero bytes read or `kPGPError_SocketsNotConnected`

A *writable* socket is one which:

- has completed a non-blocking connect
- will complete a send or sendto without blocking (the duration of this state is *not* guaranteed)

A socket having available *out-of-band data* or an outstanding *error condition* is one which

- has available out-of-band data
- has failed a non-blocking connect

- is a stream socket whose connection has been broken by its peer or a `KEEPALIVE` failure
- has an outstanding error condition that may be obtained via `PGPGetLastSocketsError`

Send Functions

PGPSend

Sends the specified data on the specified socket (which *must* be connected), and returns the number of bytes actually sent.

Syntax

```
PGPInt32 PGPSend(
             PGPSocketRef socketRef,
             const void *buffer,
             PGPInt32 bufferLength,
             PGPInt32 flags );
```

Parameters

`socketRef`	the target socket
`buffer`	the data to be sent
`bufferLength`	the length of the data to be sent
`flags`	the send flags

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPWrite

Writes the specified data on the specified socket.

Syntax

```
PGPInt32 PGPWrite(
             PGPSocketRef socketRef,
             const void *buffer,
             PGPInt32 bufferLength );
```

Parameters

socketRef	the target socket
buffer	the data to be sent
bufferLength	the length of the data to be sent

Notes

If the return value is kPGPSockets_Error, then the caller should obtain the actual error code via PGPGetLastSocketError.

PGPSendTo

For datagram sockets, sends the specified data on the specified socket, usually to the specified optional address.

For stream sockets, the optional address arguments are ignored, and so this function is equivalent to PGPSend.

In each case, the function returns the number of bytes actually sent.

Syntax

```
PGPInt32 PGPSendTo(
             PGPSocketRef socketRef,
             const void *buffer,
             PGPInt32 bufferLength,
             PGPInt32 flags,
             PGPSocketAddress *address,
             PGPInt32 addressLength );
```

Parameters

socketRef	the target socket
buffer	the data to be sent
bufferLength	the length of the data to be sent
flags	the send flags
address	the send-to address (optional)
addressLength	the length of the send-to address (optional)

Notes

If the return value is kPGPSockets_Error, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

Receive Functions

PGPReceive

Receives data on the the specified socket into the specified buffer, and returns the number of bytes actually received.

Syntax

```
PGPInt32 PGPReceive(
            PGPSocketRef socketRef,
            void *buffer,
            PGPInt32 bufferSize,
            PGPInt32 flags );
```

Parameters

`socketRef`	the target socket
`buffer`	the receiving buffer
`bufferLength`	the maximum length of the data that can be received
`flags`	the receive flags

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPRead

Reads data on the specified socket into the specified buffer, and returns the number of bytes actually read.

Syntax

```
PGPInt32 PGPRead(
            PGPSocketRef socketRef,
            void *buffer,
            PGPInt32 bufferSize );
```

Parameters

`socketRef`	the target socket
`buffer`	the receiving buffer
`bufferLength`	the maximum length of the data that can be read

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPReceiveFrom

For datagram sockets, receives data on the specified socket into the specified data buffer, usually from the specified optional address.

For stream sockets, the optional address arguments are ignored, and so this function is equivalent to `PGPreceive`.

In each case, the function returns the number of bytes actually received.

Syntax

```
PGPInt32 PGPReceiveFrom(
             PGPSocketRef socketRef,
             void *buffer,
             PGPInt32 bufferSize,
             PGPInt32 flags,
             PGPSocketAddress *address,
             PGPInt32 *addressLength );
```

Parameters

`socketRef`	the target socket
`buffer`	the receiving buffer
`bufferLength`	the maximum length of the data that can be received
`flags`	the receive flags
`address`	the receive-from address (optional)
`addressLength`	the length of the receive-from address (optional)

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

DNS and Protocol Services Functions

PGPGetHostName

Obtains the host name of the machine on which the calling application is executing.

Syntax

```
PGPInt32 PGPGetHostName(
             char *name,
             PGPInt32 nameLength );
```

Parameters

`name`	the receiving field for the target host's name
`nameLength`	the maximum length of the host name that can be received

PGPGetHostByName

Obtains the host entry for the specified host name. If no host having the specified name can be found, then the function returns NULL.

Syntax

```
PGPHostEntry * PGPGetHostByName( const char *name );
```

Parameters

name	the target host's name

PGPGetHostByAddress

Obtains the host entry for the host associated with the specified address. If no host having the specified address and type can be found, then the function returns NULL.

Syntax

```
PGPHostEntry * PGPGetHostByAddress(
             const char *address,
             PGPInt32 addressLength,
             PGPInt32 type );
```

Parameters

address	the target host's address
addressLength	the length of the target host's address (in bytes)
type	the type of the target host's address

PGPGetProtocolByName

Obtains the protocol entry for the specified protocol name. If no protocol having the specified name can be found, then the function returns NULL.

Syntax

```
PGPProtocolEntry * PGPGetProtocolByName(
             const char *name );
```

Parameters

name	the target protocol's name

PGPGetProtocolByNumber

Obtains the protocol entry for the specified protocol number. If no protocol having the specified number can be found, then the function returns NULL.

Syntax

```
PGPProtocolEntry * PGPGetProtocolByNumber(
```

```
PGPInt32 num );
```

Parameters

`num`	the target protocol's number

PGPGetServiceByName

Obtains the service entry for the specified service name. If no service having the specified name can be found, then the function returns `NULL`.

Syntax

```
PGPServiceEntry * PGPGetServiceByName(
             const char *name,
             const char *protocol );
```

Parameters

`name`	the target service's name

PGPGetServiceByPort

Obtains the service entry for the specified port / protocol combination. If the specified protocol / port combination cannot be found, then the function returns `NULL`.

Syntax

```
PGPServiceEntry * PGPGetServiceByPort(
             PGPInt32 port,
             const char *protocol );
```

Parameters

`port`	the target port
`protocol`	the protocol of the target port

Net Byte Ordering Macros

Windows & UNIX Platforms Net Byte Ordering Macros

```
PGPInt32     PGPHostToNetLong( PGPInt32 x );
PGPInt16     PGPHostToNetShort( PGPInt16 x );
PGPInt32     PGPNetToHostLong( PGPInt32 x );
PGPInt16     PGPNetToHostShort( PGPInt16 x );
```

MacOS Platforms Net Byte Ordering Macros

```
#define PGPHostToNetLong( x )( x )
#define PGPHostToNetShort( x )( x )
#define PGPNetToHostLong( x )( x )
#define PGPNetToHostShort( x )( x )
```

Error Reporting Functions

PGPGetLastSocketsError

Obtains the error number of the last function performed on the currently selected sockets (see `PGPSelect`).

Syntax

```
PGPError PGPGetLastSocketsError( void );
```

Utility Functions

PGPGetSocketName

Obtains the name associated with the specified socket, and returns its length.

Syntax

```
PGPInt32 PGPGetSocketName(
             PGPSocketRef socketRef,
             PGPSocketAddress *name,
             PGPInt32 *nameLength );
```

Parameters

`socketRef`	the target socket
`address`	the receiving field for the socket's name
`addressLength`	the length of the socket's name

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPGetPeerName

Obtains the peer name for the specified socket, and returns its length.

Syntax

```
PGPInt32 PGPGetPeerName(
             PGPSocketRef socketRef,
             PGPSocketAddress *name,
             PGPInt32 *nameLength );
```

Parameters

`socketRef`	the target socket
`address`	the receiving field for the name of the target socket's peer
`addressLength`	the length of the socket's peer's name

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPDottedToInternetAddress

Returns the numeric representation of the specified dotted string address, for example the dotted string address `127.127.127.127` would yield the numeric address `0x7F7F7F7F`.

Syntax

```
PGPUInt32 PGPDottedToInternetAddress(
             const char *address );
```

Parameters

`address`	the target Internet address, which is a *C* string of the form `255.255.255.255`

Notes

The dotted string must be `NUL` terminated.

PGPInternetAddressToDottedString

Returns the dotted string representation of the specified numeric address, for example the numeric address `0x7F7F7F7F` would yield the dotted string address `127.127.127.127`.

Syntax

```
char * PGPInternetAddressToDottedString(
             PGPInternetAddress address );
```

Parameters

`address`	the target Internet address, which is expected to be a numeric value in host byte order

Notes

The resultant dotted string Internet address is guaranteed to be `NUL` terminated.

The caller is responsible for de-allocating the resultant dotted string Internet address with `PGPFreeData`.

Control and Options Functions

PGPIOControlSocket

Sends the specified I/O control command to the specified socket.

Syntax

```
PGPInt32 PGPIOControlSocket(
             PGPSocketRef socketRef,
             PGPInt32 command,
             PGPUInt32 *commandArg );
```

Parameters

`socketRef`	the target socket
`command`	the desired I/O control command
`commandArg`	the desired I/O control command argument value

Notes

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPSetSocketOptions

Sets the specified option for the specified socket.

Syntax

```
PGPInt32 PGPSetSocketOptions(
             PGPSocketRef socketRef,
             PGPInt32 level,
             PGPInt32 optionName,
             const char *optionValue,
             PGPInt32 optionLength );
```

Parameters

`socketRef`	the target socket
`level`	the level at which the option is defined
`optionName`	the desired socket option
`optionValue`	the value of the desired socket option
`optionLength`	the length of the value of the desired socket option

Notes

For boolean options, a non-zero value is considered `TRUE`; a zero value is considered `FALSE`.

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

PGPGetSocketOptions

Obtains the specified option for the specified socket.

Syntax

```
PGPInt32 PGPGetSocketOptions(
             PGPSocketRef socketRef,
             PGPInt32 level,
             PGPInt32 optionName,
             char *optionValue,
             PGPInt32 *optionLength );
```

Parameters

`socketRef`	the target socket
`level`	the level at which the option is defined
`optionName`	the socket option to obtain
`optionValue`	the receiving field for the desired socket option
`optionLength`	the maximum length of the receiving field

Notes

For boolean options, a non-zero value is considered `TRUE`; a zero value is considered `FALSE`.

If the return value is `kPGPSockets_Error`, then the caller should obtain the actual error code via `PGPGetLastSocketError`.

TLS-related Functions

PGPSocketsEstablishTLSSession

Associates the specified socket with the specified TLS session, thus securing communications over that socket.

Syntax

```
PGPError PGPSocketsEstablishTLSSession(
            PGPSocketRef socketRef,
            PGPtlsSessionRef tlsSession );
```

Parameters

`socketRef`	the target socket
`tlsSession`	the target TLS session

BigNum Functions 13

Introduction

Modern encryption algorithms are based upon large, difficult-to-factor numbers, which in turn are based upon large primes. The PGPsdk BigNum ("*Big Number*") functions allow sophisticated PGPsdk developers access to the underlying functions that form the basis for strong cryptographic key generation. These include:

- create, copy, and free BigNum data types
- perform arithmetic operations with BigNums as operands
- perform arithmetic operations with BigNums and unsigned 16-bit quantities as operands

Many of the function descriptions include conceptual, pseudo-code examples that illustrate their processing in terms of *C* language operators and standard math library functions. However, these examples do *not* necessarily reflect either the implementation strategy or the actual usage of the function. They are *not* intended as actual sample code!

All BigNum values are considered to be non-negative, and so none of the BigNum functions will ever yield a negative result. Furthermore, most of the BigNum functions return one of:

- `kPGPError_NoErr`
- `kPGPError_BadParams`
- `kPGPError_OutOfMemory`

In most error instances, input operand values are preserved while output operand values are undefined. Notable exceptions include:

- subtraction underflow, for example:
  `( a – b )` where `|a| < |b|`)
- inversion where the number is not relatively prime to the modulus, for example:
  `gcd( x, m ) != 1`)
- divide by zero
- illegal operand overlap

The later two exceptions result in run-time assertion failures. Subtraction underflow returns `kPGPError_NoErr`, but sets its output operand value to `( b – a )`, and its underflow indicator to `TRUE`.

Header Files

```
pgpBigNum.h
```

BigNum Management Functions

PGPNewBigNum

Creates a new BigNum.

Syntax

```
PGPError PGPNewBigNum(
             PGPMemoryMgrRef pgpMemoryMgr,
             PGPBoolean useSecureMem,
             PGPBigNumRef *bn );
```

Parameters

`pgpMemoryMgr`	the target memory manager
`useSecureMem`	`TRUE` if the the resultant BigNum should be allocated in secure memory (see `PGPNewSecureData`)
`bn`	the receiving field for the resultant BigNum

Notes

The caller is responsible for deallocating the resultant BigNum with `PGPFreeBigNum`.

PGPCopyBigNum

Creates an exact copy of the specified BigNum, including its value. If the specified BigNum was allocated in secure memory, then its copy will be allocated in secure memory.

Syntax

```
PGPError PGPCopyBigNum(
             PGPBigNumRef bnOrig,
             PGPBigNumRef *bnCopy );
```

Parameters

`bnOrig`	the source BigNum
`bnCopy`	the receiving field for the copy of the BigNum

Notes

The caller is responsible for deallocating the resultant BigNum copy with `PGPFreeBigNum`.

Currently, details of the BigNum data type (specifically whether or not it resides in secure memory) are not visible at the PGPsdk level.

PGPFreeBigNum

Frees the specified BigNum.

Syntax

```
PGPError PGPFreeBigNum( PGPBigNumRef bn );
```

Parameters

`bn`	the target BigNum

Notes

BigNums do *not* have associated reference counts – the data item is always deallocated.

PGPPreallocateBigNum

Ensures that the specified BigNum can accommodate values whose expression requires at most the specified number of bits. If an error occurs, then the specified BigNum is unaltered.

Syntax

```
PGPError PGPPreallocateBigNum(
             PGPBigNumRef bn,
             PGPUInt32 numBits );
```

Parameters

`bn`	the target BigNum
`numBits`	the maximum number of bits required to express the anticipated value(s)

BigNum Assignment Functions

PGPAssignBigNum

Assigns the value of the specified source BigNum to that of the specified destination BigNum. This function differs from `PGPCopyBigNum` in that the destination BigNum must already exist.

Syntax

```
PGPError PGPAssignBigNum(
             PGPBigNumRef bnSrc,
             PGPBigNumRef bnDest );
```

Parameters

`bnSrc`	the source BigNum
`bnDest`	the destination BigNum

Notes

If the destination cannot accommodate the source value's number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPSwapBigNum

Swaps the values of two BigNums. Conceptually, this operation can be expressed as:

```
bnTmp = bn2;
bn2 = bn1;
bn1 = bnTmp;
```

Syntax

```
PGPError PGPSwapBigNum(
             PGPBigNumRef bn1,
             PGPBigNumRef bn2 );
```

Parameters

`bn1`	the first BigNum
`bn2`	the second BigNum

Notes

The source and destination are automatically resized as required (see `PGPPreallocateBigNum`).

PGPBigNumExtractBigEndianBytes

Extracts the specified number of bytes from the specified BigNum (starting at the specified offset), and places them into the specified destination buffer in big-endian order as a base 256 value, that is,

```
( bn / pow( 256, lsByte ) ) % pow( 256, numBytes )
```

Unused high-order (leading) bytes are filled with zeroes.

Syntax

```
PGPError PGPBigNumExtractBigEndianBytes(
             PGPBigNumRef bn,
             PGPByte *destBuffer,
             PGPUInt32 lsByte,
             PGPUInt32 numBytes );
```

Parameters

`bn`	the target BigNum
`destBuffer`	the receiving field for the to-be-extracted bytes, whose size must be at least `numBytes` bytes
`lsByte`	the offset (zero-based) of the starting byte
`numBytes`	the number of bytes to extract

PGPBigNumInsertBigEndianBytes

Inserts the specified number of bytes (assumed to be in big-endian order) as a base 256 value, that is,

```
( bn / pow( 256, lsByte ) ) % pow( 256, numBytes )
```

from the specified source buffer into the specified BigNum starting at the specified offset.

Syntax

```
PGPError PGPBigNumInsertBigEndianBytes(
             PGPBigNumRef bn,
             PGPByte const *srcBuffer,
             PGPUInt32 lsByte,
             PGPUInt32 numBytes );
```

Parameters

`bn`	the target BigNum
`srcBuffer`	the source field for the to-be-inserted bytes, whose size must be at least `numBytes` bytes
`lsByte`	the offset (zero-based) of the starting byte
`numBytes`	the number of bytes to insert

PGPBigNumExtractLittleEndianBytes

Extracts the specified number of bytes from the specified BigNum (starting at the specified offset), and places them into the specified destination buffer in little-endian order as a base 256 value, that is,

```
( bn / pow( 256, lsByte ) ) % pow( 256, numBytes )
```

Unused high-order (trailing) bytes are filled with zeroes.

Syntax

```
PGPError PGPBigNumExtractLittleEndianBytes(
             PGPBigNumRef bn,
             PGPByte *destBuffer,
             PGPUInt32 lsByte,
             PGPUInt32 numBytes );
```

Parameters

`bn`	the target BigNum
`destBuffer`	the receiving field for the to-be-extracted bytes, whose size must be at least `numBytes` bytes
`lsByte`	the offset (zero-based) of the starting byte
`numBytes`	the number of bytes to extract

PGPBigNumInsertLittleEndianBytes

Inserts the specified number of bytes (assumed to be in little-endian order) as a base 256 value, that is,

```
( bn / pow( 256, lsByte ) ) % pow( 256, numBytes )
```

from the specified source buffer into the specified BigNum starting at the specified offset.

Syntax

```
PGPError PGPBigNumInsertLittleEndianBytes(
             PGPBigNumRef bn,
             PGPByte const *srcBuffer,
             PGPUInt32 lsByte,
             PGPUInt32 numBytes );
```

Parameters

`bn`	the target BigNum
`srcBuffer`	the source field for the to-be-inserted bytes, whose size must be at least `numBytes` bytes
`lsByte`	the offset (zero-based) of the starting byte
`numBytes`	the number of bytes to insert

PGPBigNumGetLSWord

Returns the least significant 16 bits of the specified BigNum. If the specified BigNum has less than 16 significant bits, then the returned value is padded out with zeroes.

Syntax

```
PGPUInt16 PGPBigNumGetLSWord( PGPBigNumRef bn );
```

Parameters

`bn`	the target BigNum

PGPBigNumGetSignificantBits

Returns the number of significant bits in the specified BigNum. This will either be zero, or a value that is conceptually computed as:

```
floor( log2( bn ) ) + 1;
```

Syntax

```
PGPUInt32 PGPBigNumGetSignificantBits( PGPBigNumRef bn );
```

Parameters

`bn`	the target BigNum

BigNum Arithmetic Functions

PGPBigNumAdd

Adds the specified source BigNums, and places the result into the specified destination BigNum.

Syntax

```
PGPError PGPBigNumAdd(
             PGPBigNumRef bnSrc1,
             PGPBigNumRef bnSrc2,
             PGPBigNumRef bnDest );
```

Parameters

`bnSrc1`	the first source BigNum
`bnSrc2`	the second source BigNum
`bnDest`	the destination BigNum

Notes

Either of the source BigNums may refer to the same data item as the destination BigNum, and doing so will enhance performance.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumSubtract

Subtracts the specified second source BigNum from the specified first BigNum, and places the result into the specified destination BigNum.

Syntax

```
PGPError PGPBigNumSubtract(
            PGPBigNumRef bnSrc1,
            PGPBigNumRef bnSrc2,
            PGPBigNumRef bnDest,
            PGPBoolean *underflowInd );
```

Parameters

`bnSrc1`	the first source BigNum
`bnSrc2`	the second source BigNum
`bnDest`	the destination BigNum
`underflowInd`	`TRUE` if underflow occurred, that is (`bnSrc1 < bnSrc2`); `FALSE` otherwise.

Notes

If the source BigNums refer to the same data item, then the following is a much faster alternative:

```
PGPBigNumSetQ( bnDest, ( PGPUInt16 )0 );
```

If the first source BigNum refers to the same data item as the destination BigNum, then this will enhance performance; if the second source BigNum refers to the same data item as the destination BigNum, then this will adversely affect performance.

If the first source value is less than the second source value (subtraction underflow), then no error is returned, `underflowInd` is set to `TRUE` , and the destination value is computed by subtracting the first source BigNum from the second source BigNum, that is

```
if  ( underflowInd != (PGPBoolean *)NULL )
{
    *underflowInd = FALSE;
}
if  ( bnSrc1 < bnSrc2 )
{
    bnDest = bnSrc2 - bnSrc;
    if  ( underflowInd != (PGPBoolean *)NULL )
    {
        *underflowInd = TRUE;
    }
    err = kPGPError_NoErr;
}
else
{
    bnDest = bnSrc1 – bnSrc2;
    err = kPGPError_NoErr;
}
return( err );
```

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumCompare

Compares the specified values, and returns –`1`, `0`, or `1` depending on whether or not `bn1` is less than, equal to, or greater than `bn2`.

Syntax

```
PGPInt32 PGPBigNumCompare(
                PGPBigNumRef bn1,
                PGPBigNumRef bn2 );
```

Parameters

`bn1`	the first BigNum
`bn2`	the second BigNum

PGPBigNumSquare

Squares the specified source value, and sets the destination value to the result.

Syntax

```
PGPError PGPBigNumSquare(
             PGPBigNumRef bnSrc,
             PGPBigNumRef bnDest );
```

Parameters

`bnSrc`	the source BigNum
`bnDest`	the destination BigNum

Notes

While the source BigNum may refer to the same data item as the destination BigNum, doing so will adversely affect performance.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumMultiply

Multiplies the specified source values, and sets the destination value to the result.

Syntax

```
PGPError PGPBigNumMultiply(
             PGPBigNumRef bnMultiplicand,
             PGPBigNumRef bnMultiplier,
             PGPBigNumRef bnProduct );
```

Parameters

`bnMultiplicand`	the first source BigNum
`bnMultiplier`	the second source BigNum
`bnProduct`	the destination BigNum

Notes

While either of the source BigNums may refer to the same data item as the destination BigNum, doing so will adversely affect performance.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumDivide

Divides the specified source values, and sets the specified destination values to the resultant quotient and remainder values.

Syntax

```
PGPError PGPBigNumDivide(
             PGPBigNumRef bnNumerator,
             PGPBigNumRef bnDenominator,
             PGPBigNumRef bnQuotient,
             PGPBigNumRef bnRemainder );
```

Parameters

`bnNumerator`	the first source BigNum (numerator)
`bnDenominator`	the second source BigNum (denominator)
`bnQuotient`	the first destination BigNum (quotient)
`bnRemainder`	the second destination BigNum (remainder)

Notes

The quotient may *not* refer to the same data item as either the numerator or the denominator.

The remainder may *not* refer to the same data item as the denominator.

If the numerator and denominator refer to the same data item or have the same value, then the following is a much faster alternative:

```
PGPBigNumSetQ( bnQuotient, ( PGPUInt16 )1 );
PGPBigNumSetQ( bnRemainder, ( PGPUInt16 )0 );
```

Re-entrancy issue: the denominator is modified during the course of processing, but is restored to its original value prior to return.

The quotient and remainder are resized as required (see `PGPPreallocateBigNum`).

PGPBigNumMod

Computes the remainder that results from dividing the two source BigNums. Conceptually, this is the same as calling `PGPBigNumDivide` and ignoring the resultant quotient.

Syntax

```
PGPError PGPBigNumMod(
             PGPBigNumRef bnNumerator,
             PGPBigNumRef bnDenominator,
             PGPBigNumRef bnRemainder );
```

Parameters

`bnNumerator`	the first source BigNum (numerator)
`bnDenominator`	the second source BigNum (denomiator)
`bnRemainder`	the destination BigNum

Notes

The denominator may *not* refer to the same data item as the remainder.

If the numerator and denominator refer to the same data item or have the same value, then the following is a much faster alternative:

```
PGPBigNumSetQ( bnDest, ( PGPUInt16 )0 );
```

Re-entrancy issue: the denominator is modified during the course of processing, but is restored to its original value prior to return.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumExpMod

Raises the specified source value to the specified power, divides the intermediate value by the denominator value, and then places the remainder of the division into the destination BigNum. Conceptually, this operation can be expressed as:

```
bnRemainder = pow( bnNumerator, bnExp ) % bnDenominator;
```

Syntax

```
PGPError PGPBigNumExpMod(
            PGPBigNumRef bnNumerator,
            PGPBigNumRef bnExp,
            PGPBigNumRef bnDenominator,
            PGPBigNumRef bnRemainder );
```

Parameters

`bnNumerator`	the source BigNum
`bnExp`	the exponent BigNum
`bnDenominator`	the denominator BigNum
`bnRemainder`	the destination BigNum

Notes

The denominator *must* be odd.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumDoubleExpMod

Raises each of the source values to their associated powers, multiplies the two intermediate values, divides that intermediate value by the denominator value, and then places the remainder of the division into the destination value. Conceptually, this operation can be expressed as:

```
bnRemainder = ( pow( bnNumerator1, bnExp1 ) *
        pow( bnNumerator2, bnExp2 ) ) % bnDenominator;
```

Syntax

```
PGPError PGPBigNumDoubleExpMod(
            PGPBigNumRef bnNumerator1,
            PGPBigNumRef bnExp1,
            PGPBigNumRef bnNumerator2,
            PGPBigNumRef bnExp2,
            PGPBigNumRef bnDenominator,
            PGPBigNumRef bnRemainder );
```

Parameters

`bnNumerator1`	the first source BigNum
`bnExp1`	the first exponent BigNum
`bnNumerator2`	the second source BigNum
`bnExp2`	the second exponent BigNum
`bnDenominator`	the denominator BigNum
`bnRemainder`	the destination BigNum

Notes

The denominator *must* be odd.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumTwoExpMod

Raises two to the specified power, and then sets the destination value to the modulus of the result.

Syntax

```
PGPError PGPBigNumTwoExpMod(
            PGPBigNumRef bnExp,
            PGPBigNumRef bnDenominator,
            PGPBigNumRef bnModulus );
```

Parameters

`bnExp`	the exponent BigNum
`bnDenominator`	the modulo BigNum
`bnModulus`	the destination BigNum

Notes

The denominator *must* be odd.

This operation is equivalent to `PGPBigNumExpMod` where the numerator has a value of two.

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumInv

Divides the value 1 by the specified source, divides the intermediate value by the denominator value, and then places the remainder of the division into the destination BigNum. Conceptually, this operation can be expressed as:

```
bnRemainder = ( 1 / bnSource ) % bnDenominator;
```

Syntax

```
PGPError PGPBigNumInv(
            PGPBigNumRef bnSource,
            PGPBigNumRef bnDenominator,
            PGPBigNumRef bnRemainder );
```

Parameters

`bnSource`	the source BigNum
`bnDenominator`	the denominator BigNum
`bnRemainder`	the destination BigNum

Notes

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumLeftShift

Shifts the specified BigNum left by the specified number of bits. Vacated bit positions are zero-filled (logical shift). Conceptually, this operation can be expressed as:

```
bn = bn * pow( 2, magnitude );
```

Syntax

```
PGPError PGPBigNumLeftShift(
            PGPBigNumRef bn,
            PGPUInt32 magnitude );
```

Parameters

`bn`	the target BigNum
`magnitude`	the number of bits to shift

Notes

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumRightShift

Shifts the specified BigNum right by the specified number of bits. Vacated bit positions are zero-filled; shifted-out bits are discarded. Conceptually, this operation can be expressed as:

```
bn = floor( bn / pow( 2, magnitude ) );
```

Syntax

```
PGPError PGPBigNumRightShift(
            PGPBigNumRef bn,
            PGPUInt32 magnitude );
```

Parameters

`bn`	the target BigNum
`magnitude`	the number of bits to shift

PGPBigNumGCD

Determines the greatest common denominator for the specified source values, and places the result in the specified destination. Conceptually, this operation can be expressed as:

```
bnDest = gcd( bn1, bn2 );
```

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

Syntax

```
PGPError PGPBigNumGCD(
            PGPBigNumRef bn1,
            PGPBigNumRef bn2,
            PGPBigNumRef bnDest );
```

Parameters

`bn1`	the first BigNum
`bn2`	the second BigNum
`bnDest`	the destination BigNum

PGPBigNumMakeOdd

Determines the largest power of two that may divide the specified BigNum while yielding a resultant quotient that is greater than zero. Once determined, the specified BigNum is divided by that power of two, and the associated power itself is returned. Conceptually, this operation can be expressed as:

```
exp = 0;
while ( ( bn1 >> 1 ) > 0 )
{
    exp++;
}
bnDest = bnDest >> exp;
return( exp );
```

Syntax

```
PGPUInt16 PGPBigNumMakeOdd( PGPBigNumRef bn );
```

Parameters

`bn`	the target BigNum

Notes

The source BigNum is never expected to have a value of zero.

The resultant exponent value is never expected to exceed the maximum value of a `PGPUInt16`.

The function call:

```
err = PGPBigNumLeftShift( bn, PGPBigNumMakeOdd( bn ) );
```

is an identity operation.

BigNum 16-bit Constant Arithmetic Functions

PGPBigNumSetQ

Assigns the specified 16-bit constant as the value of the specified destination BigNum.

Syntax

```
PGPError PGPBigNumSetQ(
            PGPBigNumRef bn,
            PGPUInt16 kUInt16 );
```

Parameters

`bn`	the target BigNum
`kUInt16`	the desired 16-bit constant

Notes

The PGPsdk developer must employ additional `PGPBigNum`... functions to set a BigNum to a value greater than the maximum value of a `PGPUInt16`. These may include the arithmetic functions and/or the `PGPBigNumInsert`...`Bytes` functions.

PGPBigNumAddQ

Adds the specified 16-bit constant to the specified source value, and sets the destination value to the result.

Syntax

```
PGPError PGPBigNumAddQ(
             PGPBigNumRef bnSrc,
             PGPUInt16 kUInt16,
             PGPBigNumRef bnDest );
```

Parameters

`bnSrc`	the source BigNum
`kUInt16`	the desired 16-bit constant
`bnDest`	the destination BigNum

Notes

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumSubtractQ

Subtracts the specified 16-bit value from the specified source value, and sets the destination value to the result.

Syntax

```
PGPError PGPBigNumSubtractQ(
             PGPBigNumRef bnSrc,
             PGPUInt16 kUInt16,
             PGPBigNumRef bnDest,
             PGPBoolean *underflowInd );
```

Parameters

bnSrc	the source BigNum
kUInt16	the desired 16-bit constant
bnDest	the destination BigNum
underflowInd	TRUE if underflow occurred, that is (bnSrc1 < kUInt16); FALSE otherwise.

Notes

If the source value is less than the 16-bit constant value, then no error is returned, underflowInd is set to TRUE, and the destination value is computed by subtracting the source BigNum from the 16-bit constant, that is

```
if  ( underflowInd != (PGPBoolean *)NULL )
{
    *underflowInd = FALSE;
}
if  ( bnSrc < kUInt16 )
{
    bnDest = kUInt16 - bnSrc;
    if  ( underflowInd != (PGPBoolean *)NULL )
    {
        *underflowInd = TRUE;
    }
}
else
{
    bnDest = bnSrc - kUInt16;
}
return( kPGPError_NoError );
```

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see PGPPreallocateBigNum).

PGPBigNumCompareQ

Compares the value of the specified BigNum with that of the specified 16-bit constant, and returns –1, 0, or 1 depending on whether or not bn is less than, equal to, or greater than the specified 16-bit constant.

Syntax

```
PGPInt32 PGPBigNumCompareQ(
            PGPBigNumRef bn,
            PGPUInt16 kUInt16 );
```

Parameters

`bn`	the target BigNum
`kUInt16`	the desired 16-bit constant

PGPBigNumMultiplyQ

Multiplies the specified source value by the specified 16-bit constant, and sets the destination value to the result.

Syntax

```
PGPError PGPBigNumMultiplyQ(
             PGPBigNumRef bnSrc,
             PGPUInt16 kUInt16,
             PGPBigNumRef bnDest );
```

Parameters

`bnSrc`	the source BigNum
`kUInt16`	the desired 16-bit constant
`bnDest`	the destination BigNum

Notes

If the destination cannot accommodate the resultant number of significant bits, then the destination is automatically resized (see `PGPPreallocateBigNum`).

PGPBigNumModQ

Computes the modulus of the specified values, and returns the result. Conceptually, this operation can be expressed as:

```
return( bnNumerator % bnDenominator );
```

Syntax

```
PGPUInt16 PGPBigNumModQ(
             PGPBigNumRef bnNumerator,
             PGPUInt16 bnDenominator );
```

Parameters

`bnNumerator`	the source BigNum
`bnDenominator`	the desired modulo

PGPsdk Error Summary

A

Introduction

The PGPsdk functions return a large number of error codes, and these are both enumerated and explained in this appendix. However, the PGPsdk developer should keep the following points in mind when making use of this information:

- the listed error codes and their related descriptions are specific to this instance of the PGPsdk only (Version 1.7.1), and are subject to change in later instances
- the circumstances under which a particular error code is returned are subject to change in a later instance of the PGPsdk
- a particular error code may be superseded by another and / or more specific error code in a later instance of the PGPsdk
- several error codes are currently unused and / or unimplemented. These and possibly other error codes may be removed from a later instance of the PGPsdk
- a particular error codes may not currently be visible at the PGPsdk level, or may not currently be visible under certain circumstances. For example, the PGPsdk “convenience” functions may supersede a specific lower-level error code with a more general error code

Table A-1. Generic Errors

Generic Error Constant
kPGPError_NoErr
kPGPError_BadParams
kPGPError_BadPassphrase
kPGPError_BufferTooSmall
kPGPError_CorruptData
kPGPError_EndOfIteration
kPGPError_FeatureNotAvailable
kPGPError_ImproperInitialization
kPGPError_IncompatibleAPI
kPGPError_ItemAlreadyExists
kPGPError_ItemNotFound
kPGPError_LazyProgrammer
kPGPError_OptionNotFound
kPGPError_OutOfEntropy
kPGPError_OutOfMemory
kPGPError_PrefNotFound
kPGPError_RedundantOptions
kPGPError_UnknownError
kPGPError_UnknownRequest
kPGPError_UserAbort

Table A-2. File Errors

File Error Constant
kPGPError_CantOpenFile
kPGPError_DiskFull
kPGPError_DiskLocked
kPGPError_EOF
kPGPError_FileCorrupt
kPGPError_FileLocked
kPGPError_FileNotFound
kPGPError_FileOpFailed
kPGPError_FilePermissions
kPGPError_IllegalFileOp
kPGPError_NoMacBinaryTranslationAvailable
kPGPError_NotMacBinary
kPGPError_ReadFailed
kPGPError_WriteFailed

Table A-3. Keyring Validity Check Errors

Keyring Validity Error Constant
kPGPError_AdditionalRecipientRequestKeyNotFound
kPGPError_BadPacket
kPGPError_TroubleBadTrust
kPGPError_TroubleBareKey
kPGPError_TroubleDuplicateKey
kPGPError_TroubleDuplicateKeyID
kPGPError_TroubleDuplicateName
kPGPError_TroubleDuplicateSecretKey
kPGPError_TroubleDuplicateSignature
kPGPError_TroubleDuplicateUnknown
kPGPError_TroubleImportingNonexportableSignature
kPGPError_TroubleKeySubKey
kPGPError_TroubleKeyTooBig
kPGPError_TroubleNameTooBig
kPGPError_TroubleNewSecretKey
kPGPError_TroubleOldSecretKey
kPGPError_TroubleSecretKeyTooBig
kPGPError_TroubleSignatureTooBig
kPGPError_TroubleSigSubKey
kPGPError_TroubleUnexpectedName
kPGPError_TroubleUnexpectedSignature
kPGPError_TroubleUnexpectedSubKey
kPGPError_TroubleUnexpectedTrust
kPGPError_TroubleUnexpectedUnknown
kPGPError_TroubleUnknownPacketByte
kPGPError_TroubleUnknownTooBig
kPGPError_TroubleVersionBugCur
kPGPError_TroubleVersionBugPrev

Table A-4. Key Set Filter Errors

Filter Error Constant
kPGPError_InconsistentFilterClasses
kPGPError_InvalidFilterParameter
kPGPError_UnknownFilterType
kPGPError_UnsupportedHKPFilter
kPGPError_UnsupportedLDAPFilter

Table A-5. Key, Sub-Key, and User ID Errors

Key-Related Error Constant
kPGPError_CertifyingKeyDead
kPGPError_DuplicateCert
kPGPError_DuplicateUserID
kPGPError_InvalidProperty
kPGPError_ItemIsReadOnly
kPGPError_ItemWasDeleted
kPGPError_KeyDisabled
kPGPError_KeyExpired
kPGPError_KeyInvalid
kPGPError_KEY_LONG
kPGPError_KeyPacketTruncated
kPGPError_KeyRevoked
kPGPError_KeyTooLarge
kPGPError_KeyUnusableForEncryption
kPGPError_KeyUnusableForSignature
kPGPError_MalformedKeyComponent
kPGPError_MalformedKeyExponent
kPGPError_MalformedKeyModulus
kPGPError_PublicKeyUnimplemented
kPGPError_RSAPublicExponentIsEven
kPGPError_RSAPublicModulusIsEven
kPGPError_UnknownKeyVersion
kPGPError_UnknownPublicKeyAlgorithm
kPGPError_UnknownString2Key

Table A-6. Signature Errors

Signature Error Constant
kPGPError_BadSignatureSize
kPGPError_ExtraDateOnSignature
kPGPError_ExtraSignatureMaterial
kPGPError_MalformedSignatureInteger
kPGPError_SignatureBitsWrong
kPGPError_SIG_LONG
kPGPError_TruncatedSignature
kPGPError_UnknownSignatureAlgorithm
kPGPError_UnknownSignatureType
kPGPError_UnknownSignatureVersion
kPGPError_X509InvalidCertificateFormat
kPGPError_X509InvalidCertificateSignature
kPGPError_X509NeededCertNotAvailable
kPGPError_X509SelfSignedCert

Table A-7. Encode/Decode Errors

Encode/Decode Error Constant
kPGPError_AsciiParseIncomplete
kPGPError_CombinedConventionalAndPublicEncryption
kPGPError_CorruptSessionKey
kPGPError_DetachedSignatureFound
kPGPError_DetachedSignatureWithEncryption
kPGPError_DetachedSignatureWithoutSigningKey
kPGPError_InconsistentEncryptionAlgorithms
kPGPError_InputFile
kPGPError_Interrupted
kPGPError_MissingEventHandler
kPGPError_MissingKeySet
kPGPError_MissingPassphrase
kPGPError_MultipleInputOptions
kPGPError_MultipleOutputOptions
kPGPError_NoDecryptionKeyFound
kPGPError_NoInputOptions
kPGPError_NoOutputOptions
kPGPError_OutputBufferTooSmall
kPGPError_SkipSection
kPGPError_TooManyARRKs

Table A-8. Key Server Errors

Key Server Error Constant
kPGPError_ServerAddFailed
kPGPError_ServerAuthorizationFailed
kPGPError_ServerAuthorizationRequired
kPGPError_ServerBadKeysInSearchResults
kPGPError_ServerBindFailed
kPGPError_ServerConnectFailed
kPGPError_ServerCorruptKeyBlock
kPGPError_ServerInvalidProtocol
kPGPError_ServerKeyAlreadyExists
kPGPError_ServerKeyFailedPolicy
kPGPError_ServerOpenFailed
kPGPError_ServerOperationNotAllowed
kPGPError_ServerPartialAddFailure
kPGPError_ServerRequestFailed
kPGPError_ServerSearchFailed
kPGPError_ServerSocketError
kPGPError_ServerTooManyResults
kPGPError_ServerUnknownHost
kPGPError_ServerUnknownResponse

Table A-9. Client/Server Communications Errors

Communication Error Constant
kPGPError_SocketsAddressFamilyNotSupported
kPGPError_SocketsAddressInUse
kPGPError_SocketsAddressNotAvailable
kPGPError_SocketsAlreadyConnected
kPGPError_SocketsBufferOverflow
kPGPError_SocketsDomainServerError
kPGPError_SocketsHostNotFound
kPGPError_SocketsInProgress
kPGPError_SocketsListenQueueFull
kPGPError_SocketsNetworkDown
kPGPError_TLSAlertReceived
kPGPError_TLSKeyUnusable
kPGPError_TLSNoCommonCipher
kPGPError_TLSProtocolViolation
kPGPError_TLSUnexpectedClose
kPGPError_TLSVersionUnsupported
kPGPError_TLSWrongState

Table A-10. Rarely Encountered PGP Errors

Error Constant
kPGPError_AssertFailed
kPGPError_BadCipherNumber
kPGPError_BadHashNumber
kPGPError_BadKeyLength
kPGPError_BadMemAddress
kPGPError_BadSessionKeyAlgorithm
kPGPError_BadSessionKeySize
kPGPError_BigNumNoInverse
kPGPError_CantDecrypt
kPGPError_CantHash
kPGPError_ConfigParseFailure
kPGPError_ConfigParseFailureBadFunction
kPGPError_ConfigParseFailureBadOptions
kPGPError_EnvPriorityTooLow
kPGPError_FIFOReadError
kPGPError_InvalidCommit
kPGPError_KeyIsLocked
kPGPError_OutOfRings
kPGPError_PublicKeyTooLarge
kPGPError_PublicKeyTooSmall
kPGPError_RandomSeedTooSmall
kPGPError_SecretKeyNotFound
kPGPError_SizeAdviseFailure
kPGPError_UnbalancedScope
kPGPError_UnknownCharMap
kPGPError_UnknownVersion
kPGPError_WrongScope

Generic Errors

kPGPError_NoErr

Success; no error occurred.

kPGPError_BadParams

- an invalid parameter object or parameter value was detected. This error may be superseded by a specific function- or value-related error, for example `kPGPError_InvalidFilterParameter`
- an option list contains mutually exclusive options
- an option list does not contain one or more required options

kPGPError_BadPassphrase

The indicated passphrase:

- does not unlock the associated key
- does not authorize the requested key server operation

This may be due to an incorrect passphrase, or to a passphrase having zero length. Rarely, this may indicate an internal error where an expected passphrase parameter was `NULL`.

`kPGPError_BufferTooSmall`

The indicated buffer cannot hold all of the resultant data; partial data may be present. This error applies to functions that return a one-time, discrete value, for example, `PGPGetErrorString`, and should not be confused with `kPGPError_OutputBufferTooSmall`.

`kPGPError_CorruptData`

- an Elgamal checksum did not match
- an RSA key length is invalid
- the key data is not valid for the key's version, for example, lengths and even/odd values
- the group set checksum did not match that expected

`kPGPError_EndOfIteration`

End of iteration (see the `PGPKeyIter`... functions).

`kPGPError_FeatureNotAvailable`

The requested feature, while recognized, is not available with this instance of the PGPsdk.

`kPGPError_ImproperInitialization`

- the PGPsdk has not been properly initialized (see `PGPsdkInit`)
- the cipher context has not been properly initialized (see `PGPInitSymmetricCipher`, `PGPInitCBC`, and `PGPInitCFB`)
- the in-force preferences could not be obtained from the current context (see the preference functions)

`kPGPError_IncompatibleAPI`

The underlying PGPsdk library version is too old or too new.

`kPGPError_ItemAlreadyExists`

The exact key or component already exists (see the key manipulation functions, for example `PGPAddUserID`).

`kPGPError_ItemNotFound`

- a packet, key, or component was not found (see the key manipulation functions, for example `PGPRevokeSubKey`)
- an unknown feature selector value was specified (see `PGPGetFeatureFlags`)

`kPGPError_LazyProgrammer`

- a key ring cannot be closed due to usage conflicts
- a buffer cannot be flushed because its context is not flagged as being writable

`kPGPError_OptionNotFound`

The indicated option was not found (implies that a required option was omitted), or is not valid for the indicated operation.

`kPGPError_OutOfEntropy`

The global random number pool contains insufficient random bits to:

- generate a key using the indicated public key algorithm
- encrypt a block of data to the indicated key(s)

kPGPError_OutOfMemory
Could not obtain the required amount of memory.

kPGPError_PrefNotFound
The requested preference was not found, or is not valid for the indicated object and/or operation.

kPGPError_RedundantOptions
Multiple instances of an option that may only appear once were found in the option list.

kPGPError_UnknownError
Unknown error.

kPGPError_UnknownRequest
Unrecognized request.

kPGPError_UserAbort
The user cancelled the operation. This always results from an event handler returning this error code, and its subsequent propagation to the initiating function, for example, `PGPEncode`.

File-related Errors

The exact meanings of these file-related errors may differ according to platform, particularly the exact meaning of and reason(s) for returning `kPGPError_CantOpenFile`.

kPGPError_CantOpenFile
Non-specific file open failure. This could be due to insufficient memory, exceeding a platform-specific limit, for example, too many open files, or generalization of a more specific error due to platform-specific error reporting limitations.

kPGPError_DiskFull
Cannot write to file – disk or file system is full.

kPGPError_DiskLocked
A write operation was attempted on a disk that was not flagged as being writable.

kPGPError_EOF
End of file encountered.

kPGPError_FileCorrupt
The key database is corrupt.

kPGPError_FileLocked
A write operation was attempted on a file that was not flagged as being writable.

kPGPError_FileNotFound
File not found.

kPGPError_FileOpFailed
Non-specific file operation failure. This almost always results from an underlying platform I/O error.

kPGPError_FilePermissions
- the caller has insufficient privileges to open the file in the indicated mode
- the file resides on a read-only file system

kPGPError_IllegalFileOp

The requested file operation is illegal, either from a platform perspective or a PGPsdk perspective:

- a read operation was attempted on a pipe or file that was not flagged as being readable
- an attempt was made to change a file from writable to readable on MacOS
- an attempt was made to revert an in-memory key database that has not been committed, and so does not have a current backing store

kPGPError_NoMacBinaryTranslationAvailable

Translation to Macintosh MacBinary file format is not available.

kPGPError_NotMacBinary

The indicated file is not a Macintosh MacBinary file.

kPGPError_ReadFailed

Non-specific read-from-file failure.

kPGPError_WriteFailed

Non-specific write-to-file failure.

Key Ring Validity Check Errors

These errors are returned primarily from the internal key ring open / read / merge routines during the validity check phase. If any of these errors occurs, then the key ring contains one or more invalid and / or corrupted packets, keys, or components.

The `kPGPError_Trouble`... error codes are primarily internal errors, and are almost always superseded at the PGPsdk level by a more generic error, for example, `kPGPError_BadPacket`.

kPGPError_AdditionalRecipientRequestKeyNotFound

The *referenced* additional recipient request key does not exist, that is, the key identified by the current key's additional recipient request key component does not exist. Instances where a *component* additional recipient request key does not exist reflect `kPGPError_ItemNotFound` (see `PGPGetIndexedAdditionalRecipientRequest`.

kPGPError_BadPacket

Bad packet.

kPGPError_TroubleBadTrust

Trust packet malformed.

kPGPError_TroubleBareKey

Key found with no associated User ID(s). Minimally, that of the key owner should always exist.

kPGPError_TroubleDuplicateKey

Duplicate key (in the same key ring).

kPGPError_TroubleDuplicateKeyID

Duplicate KeyID, different keys.

kPGPError_TroubleDuplicateName

Duplicate User ID (in the same key ring).

kPGPError_TroubleDuplicateSecretKey

Duplicate private key (in the same key ring).

kPGPError_TroubleDuplicateSignature
Duplicate signature (in the same key ring).

kPGPError_TroubleDuplicateUnknown
Duplicate unknown item in the key ring.

kPGPError_TroubleKeySubKey
The current key matches one of its sub-keys.

kPGPError_TroubleKeyTooBig
The current key is grossly oversized, that is, its data overflows the internal buffer, which is sized to accommodate the largest possible key.

kPGPError_TroubleNameTooBig
The current User ID is grossly oversized, that is, its data overflows the internal buffer, which is sized to accommodate the largest possible User ID.

kPGPError_TroubleNewSecretKey
Internal error – currently unimplemented.

kPGPError_TroubleOldSecretKey
Internal error – currently unimplemented.

kPGPError_TroubleSecretKeyTooBig
The current private key is grossly oversized, that is, its data overflows the internal buffer, which is sized to accommodate the largest possible private key.

kPGPError_TroubleSignatureTooBig
The current signature is grossly oversized, that is, its data overflows the internal buffer, which is sized to accommodate the largest possible signature.

kPGPError_TroubleSigSubKey
The current signature is based upon a sub-key, rather than upon a key.

kPGPError_TroubleUnexpectedName
A User ID was found that is not associated with any key.

kPGPError_TroubleUnexpectedSignature
A signature was found that is not associated with any key.

kPGPError_TroubleUnexpectedSubKey
A sub-key was found that is not associated with any key.

kPGPError_TroubleUnexpectedTrust
A trust packet was found that is not associated with any key.

kPGPError_TroubleUnexpectedUnknown
A packet of unknown type was found that is not associated with any key.

kPGPError_TroubleUnknownPacketByte
A packet of unknown type was found that is associated with a key.

kPGPError_TroubleUnknownTooBig
The current packet is of an unknown type, and its length exceeds that of the largest possible packet.

kPGPError_TroubleVersionBugPrev
Internal error related to the current private key's version.

kPGPError_TroubleVersionBugCur
Internal error related to the current private key's version.

Key Filter Errors

kPGPError_InconsistentFilterClasses

PGPIntersectFilters or PGPUnionFilters specifies filters that have incompatible filter classes. Currently, the PGPsdk defines only one filter class, and so this implies an internal PGPsdk error.

kPGPError_InvalidFilterParameter

An invalid filter function parameter value was detected, for example, PGPNewKeyEncryptAlgorithmFilter specified an invalid value for its encryptAlgorithm parameter. This differs from kPGPError_BadParams only in that it is specific to the key filter functions.

kPGPError_UnknownFilterType

Unknown filter type. This implies an internal PGPsdk error.

kPGPError_UnsupportedHKPFilter

Filter translation failed – the resultant query is not supported by HTTP key servers.

kPGPError_UnsupportedLDAPFilter

Filter translation failed – the resultant query is not supported by LDAP key servers.

Key Errors

These errors are encountered when parsing a key or sub-key packet. If multiple errors occur, then only the last error is reported. Parse errors imply corrupted packets; non-parse errors imply incorrect key or sub-key data.

kPGPError_CertifyingKeyDead

The signing key has been revoked, has expired, or is otherwise invalid.

kPGPError_DuplicateCert

Multiple signatures by the same key exist, and more than one is not revoked.

kPGPError_DuplicateUserID

Multiple User IDs of the same name exist, and more than one is not revoked.

kPGPError_InvalidProperty

The indicated key or component property is:

- invalid for the key or component, for example, key *vs*. signature
- invalid for the nature of the key or component, for example, public key *vs*. private key
- invalid for the data type of the key or component, for example, PGPGetKeyBoolean was passed the name of a numeric property (see Tables 2-4, 2-5, and 2-6)

kPGPError_ItemIsReadOnly

The indicated key or component belongs to a read-only key set

kPGPError_ItemWasDeleted

The indicated key or component has already been deleted.

kPGPError_KeyDisabled

The current key has been disabled.

kPGPError_KeyExpired

The current key has expired.

kPGPError_KeyInvalid

The current key validity is below that specified as being acceptable (see PGPOFailBelowValidity

kPGPError_KEY_LONG

Parse - warning! Key packet has extraneous trailing bytes. This implies that a valid key was found *before* encountering any extraneous data in the packet.

kPGPError_KeyPacketTruncated

Parse - the current key packet is too short.

kPGPError_KeyRevoked

The current key has been revoked.

kPGPError_KeyTooLarge

- a **DSA** key (public or private portion) exceeds the allowable size. However, when the private portion of the key is being generated and its requested length is too large, a kPGP_PublicKeyTooLarge error is recognized
- an RSA key (public or private portion) exceeds the allowable size. However, when the key is being used for encryption and its length is too large, a kPGP_PublicKeyTooLarge error is recognized.

kPGPError_KeyUnusableForEncryption

The current key cannot be used for encryption (currently unused – will reflect kPGPError_PublicKeyUnimplemented).

kPGPError_KeyUnusableForSignature

The current key cannot be used for signing (currently unused – will reflect kPGPError_PublicKeyUnimplemented).

kPGPError_MalformedKeyComponent

Parse - the current key component is badly formatted.

kPGPError_MalformedKeyExponent

Parse - the current key exponent is badly formatted.

kPGPError_MalformedKeyModulus

Parse - the current key modulus is badly formatted.

kPGPError_PublicKeyUnimplemented

The indicated public key operation is invalid, unknown, or unimplemented. This includes:

- a sub-key which is flagged as being able to both sign and encrypt
- an attempt was made to encrypt with a key which can only sign, or vice versa
- an attempt to encrypt with a DSA key, or to use DSA for an encrypted session key

kPGPError_RSAPublicModulusIsEven

The current key is an RSA public key whose modulus is even, which is not valid.

kPGPError_RSAPublicExponentIsEven

The current key is an RSA public key whose exponent is even, which is not valid.

kPGPError_UnknownKeyVersion

The version of the current key is unknown.

kPGPError_UnknownPublicKeyAlgorithm
The public key algorithm is unknown or unsupported (see PGPGetIndexedPublicKeyAlgorithmInfo. This indicates that the active key was generated with an algorithm that is not implemented for that instance of the PGPsdk. For example, passing an RSA key to any function of an Elgamal-only instance of the PGPsdk will result in this error.

kPGPError_UnknownString2Key
The format of the string representation of a key did not correspond to that of any known format, and so the string could not be converted to binary format. This implies invalid export data, or a mismatch between the PGPsdk and the PGP software which created the string.

Signature Errors

If multiple errors occur, only the last error is reported. Parse errors imply corrupted signature packets; non-parse errors imply incorrect signature data.

kPGPError_BadSignatureSize
Invalid signature – incorrect size (may be too short or too long).

kPGPError_ExtraDateOnSignature
Parse - additional signature date component(s) detected.

kPGPError_ExtraSignatureMaterial
Parse - additional unrecognized signature information detected.

kPGPError_MalformedSignatureInteger
Parse - Signature integer component improperly formatted.
Parse - Signature integer component improperly formatted.

kPGPError_SignatureBitsWrong
Invalid signature – incorrect number of bits (RSA signatures only).

kPGPError_SIG_LONG
Parse - warning! Signature packet has extraneous trailing bytes. This differs from the "extra" and too long / too short errors in that a valid signature was found *before* encountering any extraneous data in the packet.

kPGPError_TruncatedSignature
Parse - the signature data is shorter than that expected.

kPGPError_UnknownSignatureAlgorithm
Parse - unknown signature algorithm (applies only to signature versions using RSA).

kPGPError_UnknownSignatureType
The signature data indicated an unknown PGP signature type.

kPGPError_UnknownSignatureVersion
Parse - the signature data indicated an unknown PGP signature version.

kPGPError_X509InvalidCertificateFormat

- the length of the certificate is 0 (zero) bytes
- the timestamp(s) contains invalid characters
- the indicated public key algorithm is invalid or not supported
- the indicated creation time is *after* the indicated expiration time
- the data items in the certificate are not in the expected sequence
- could not create the appropriate hash context for signature verification. This may be due to an unsupported hash algorithm.

kPGPError_X509InvalidCertificateSignature

The certificate's signature failed verification.

kPGPError_X509NeededCertNotAvailable

An expected certificate could not be found. This implies a broken certificate chain.

kPGPError_X509SelfSignedCert

A child certificate was signed by its parent.

Encode/Decode Errors

kPGPError_AsciiParseIncomplete

ASCII armor input is incomplete (decode only). This implies a failed encryption, a failed transmission, or other corruption of the armored cipher text.

kPGPError_CombinedConventionalAndPublicEncryption

Invalid option combination – both conventional encryption and public key encryption were requested.

kPGPError_CorruptSessionKey

The encrypted session key is bad.

kPGPError_DetachedSignatureFound

A detached signature was found, but no event handler is defined to receive the `kPGPEventDetachedSignatureEvent` posting.

kPGPError_DetachedSignatureWithEncryption

Invalid option combination - encryption requested with a detached signature.

kPGPError_DetachedSignatureWithoutSigningKey

Invalid option combination - no signing key found for the detached signature.

kPGPError_InconsistentEncryptionAlgorithms

At least one of the recipients identified by the encrypt-to key set does not specify the same encryption algorithm as the other recipients.

kPGPError_InputFile

The indicated input file could not be opened.

kPGPError_Interrupted

Non-fatal interruption of the current operation.

kPGPError_MissingEventHandler

Event posting was requested for the operation, but no event handler is defined.

kPGPError_MissingKeySet

The key set(s) containing the available decoding key(s) was omitted from the option list.

kPGPError_MissingPassphrase

A required passphrase is missing, which usually indicates an omitted passphrase option (see PGPOPassphrase and PGPOPassphraseBuffer), but may also indicate a passphrase having zero length.

kPGPError_MultipleInputOptions

This operation accepts only a single input specification. This indicates that multiple, distinct input options were found, rather than multiple instances of the same input option (see kPGPError_RedundantOptions).

kPGPError_MultipleOutputOptions

This operation accepts only a single output specification. This indicates that multiple, distinct output options were found, rather than multiple instances of the same output option (see kPGPError_RedundantOptions).

kPGPError_NoDecryptionKeyFound

None of the keys in the indicated decryption key set(s) is capable of decoding the cipher text (decode only).

kPGPError_NoInputOptions

No input source was indicated for the requested operation.

kPGPError_NoOutputOptions

No output destination was indicated for the requested operation.

kPGPError_OutputBufferTooSmall

The PGPsdk outputs data as discrete blocks, and a resultant block is larger than the indicated buffer (see PGPOOutputBuffer). This error applies to functions that output an arbitrary amount of data, for example, PGPDecode, and should not be confused with kPGPError_BufferTooSmall.

kPGPError_SkipSection

The user requested skipping of this lexical section (decode only). This implies that the event handler returned this “error” in response to a kPGPEvent_BeginLexEvent.

kPGPError_TooManyARRKs

The additional decryption key key set contains too many keys (currently limited to four; see PGPOAdditionalRecipientRequestKeySet).

Key Server Errors

kPGPError_ServerAddFailed

Adding a specific key to the server failed. This is an internal error, and is reflected at the PGPsdk level as kPGPError_ServerPartialAddFailure.

kPGPError_ServerAuthorizationFailed

The required authorization for this operation failed. This implies that the server was not created for administrator access (see PGPNewKeyServerFromURL accessType argument).

kPGPError_ServerAuthorizationRequired

Authorization is required for this operation. This implies that the server was not created for administrator access (see PGPNewKeyServerFromURL accessType argument).

kPGPError_ServerBadKeysInSearchResults

The search results contain one or more corrupt keys.

kPGPError_ServerBindFailed
Server bind failure.

kPGPError_ServerConnectFailed
Non-specific server connect failure.

kPGPError_ServerCorruptKeyBlock
Corrupt key block – public key decode failure. This is an obsolete HTTP server error.

kPGPError_ServerInvalidProtocol
The server protocol is neither HTTP nor LDAP. Except when issued by `PGPNewKeyServerFromURL`, this should be considered an internal error.

kPGPError_ServerKeyAlreadyExists
The key being added to the server already exists on that server.

kPGPError_ServerKeyFailedPolicy
One or more keys being uploaded failed the server policy check.

kPGPError_ServerOpenFailed
Server open failed (LDAP servers only).

kPGPError_ServerOperationNotAllowed
The requested operation is not permitted for this server. This occurs most frequently for HTTP servers, which support only a limited set of operations.

kPGPError_ServerPartialAddFailure
At least one key could not be added to the server; the `PGPUploadToKeyServer` argument `keysThatFailed` will reference a non-empty key set.

kPGPError_ServerRequestFailed
The server rejected the request.

kPGPError_ServerSearchFailed
The search failed; this implies that no qualifying keys were found.

kPGPError_ServerSocketError
Non-specific socket layer error.

kPGPError_ServerTooManyResults
The search returned too many items, or exceeded the maximum time.

kPGPError_ServerUnknownHost
The specified host could not be located. This implies an incorrect host name, or a network configuration / domain look-up issue.

kPGPError_ServerUnknownResponse
The server replied with an unknown response. This implies an internal error, or a mismatch between the key server and PGPsdk versions.

Client/Server Communication Errors

kPGPError_SocketsAddressFamilyNotSupported

kPGPError_SocketsAddressInUse

kPGPError_SocketsAddressNotAvailable

kPGPError_SocketsAlreadyConnected

kPGPError_SocketsBufferOverflow

kPGPError_SocketsDomainServerError

kPGPError_SocketsHostNotFound

kPGPError_SocketsInProgress

kPGPError_SocketsListenQueueFull

kPGPError_SocketsNetworkDown

kPGPError_SocketsNotASocket

kPGPError_SocketsNotBound

kPGPError_SocketsNotConnected

kPGPError_SocketsNotInitialized

kPGPError_SocketsOperationNotSupported

kPGPError_SocketsProtocolNotSupported

kPGPError_SocketsTimedOut

kPGPError_TLSAlertReceived

A fatal error ocurred while processing a request.

kPGPError_TLSKeyUnusable

The key presented to `PGPSetLocalPrivateKey` is not secret, cannot sign, or is disabled, expired, or revoked.

kPGPError_TLSNoCommonCipher

A mutually agreeable cipher suite cannot be found.

kPGPError_TLSProtocolViolation

A data format error was detected:

- unknown packet type received
- indicated packet length is 0 (zero) or exceeds the maximum packet length
- actual packet length does not match indicated packet length
- the indicated number of cipher suites cannot fit in the actual packet length
- the packet compression method is not supported
- invalid alert data length (internal error)

An operation sequencing error was detected:

- invalid / unexpected state change request

kPGPError_TLSUnexpectedClose

A read / write operation resulted in 0 (zero) bytes being transferred.

kPGPError_TLSVersionUnsupported

The indicated server or packet version is not supported.

kPGPError_TLSWrongState

The requested operation is not valid for the current state, for example, a handshake request when not idle, or a send / receive request when not ready.

Rarely Encountered PGP Errors

These error codes should rarely be encountered, if ever. Most are indicative of internal PGPsdk errors, and not all are propagated to the PGPsdk level.

kPGPError_AssertFailed

Assertion failure; currently unimplemented. Depending upon the platform, a function that would return this error simply asserts.

kPGPError_BadCipherNumber

The implied public key algorithm is unknown, which implies an internal error (PGPsdk functions which accept an explicit cipher algorithm parameter return `kPGPError_BadParams`).

kPGPError_BadHashNumber

The implied hash algorithm is unknown, which implies an internal error (PGPsdk functions which accept an explicit hash algorithm parameter return `kPGPError_AlgorithmNotAvailable`). However, under certain circumstances, this may mask an out-of-memory condition.

kPGPError_BadKeyLength

Illegal key length for the implied algorithm.

kPGPError_BadMemAddress

Bad memory address. Unimplemented. In many cases, an invalid address (especially `NULL`) will be reflected as `kPGPError_BadParams`.

kPGPError_BadSessionKeyAlgorithm

The public key algorithm used for the encrypted session key is unknown or unsupported (see `kPGPError_UnknownPublicKeyAlgorithm`).

kPGPError_BadSessionKeySize

The indicated encrypted session key is too short.

kPGPError_BigNumNoInverse

kPGPError_CantDecrypt

Cannot decrypt message - invalid or corrupted cipher text (specifically, an initialization vector mismatch).

kPGPError_CantHash

Cannot hash message – unable to create hash list for processing signatures.

kPGPError_ConfigParseFailure

An error occurred while parsing the configuration file.

kPGPError_ConfigParseFailureBadFunction

An option indicating an unknown or unsupported function was found while parsing the configuration file. This implies an invalid configuration file, or a mismatch between the configuration file version and the instance of the PGPsdk.

kPGPError_ConfigParseFailureBadOptions

An unknown option was found while parsing the configuration file. This implies an invalid configuration file, or a mismatch between the configuration file version and the instance of the PGPsdk.

kPGPError_EnvPriorityTooLow

Environment variable not set: priority too low.

kPGPError_FIFOReadError

Incomplete read from FIFO list. This error is associated with parsing ASCII armor data, and implies that the data is corrupted or invalid, that is, not in ASCII armor format.

kPGPError_InvalidCommit

Invalid commit. This error is associated with parsing annotations included in the cipher text.

kPGPError_KeyIsLocked

- an encrypted session key cannot be unlocked due to an incorrect or missing passphrase
- a signature cannot be calculated because the required key is locked
- a key cannot be re-encrypted with a new passphrase because that key is locked, which implies an incorrect or missing old passphrase

kPGPError_OutOfRings

Internal key ring bits exhausted.

kPGPError_PublicKeyTooLarge

The indicated public key size exceeds the PGPsdk limit (limit varies by public key algorithm and type).

kPGPError_PublicKeyTooSmall

The indicated public key is too small to contain all of the indicated data (required size varies by public key algorithm and type).

kPGPError_RandomSeedTooSmall

The file specified to seed the global random number pool contains an insufficient amount data.

kPGPError_SecretKeyNotFound
No secret key found.

kPGPError_SizeAdviseFailure
`sizeAdvise` promise not kept.

kPGPError_UnbalancedScope
A nesting error was detected while parsing annotations included in the cipher text.

kPGPError_UnknownCharMap
The requested character set is unknown or not supported, so no translation to/from that character set is available.

kPGPError_UnknownVersion
The version of an encrypted session key or a signature is unknown.

kPGPError_WrongScope
Data sent in wrong scope. This error is associated with parsing annotations included in the cipher text, and implies a nesting error

Glossary

A5	a trade-secret cryptographic algorithm used in European cellular telephones.
Access control	a method of restricting access to resources, allowing only privileged entities access.
Additional recipient request key	a special key whose presence that indicates that all messages encrypted to its associated base key should also be automatically encrypted to it. Sometimes referred to by its marketing term, *additional decryption key*.
AES (Advanced Encryption Standard)	NIST approved standards, usually used for the next 20 - 30 years.
AKEP (Authentication Key Exchange Protocol)	key transport based on symmetric encryption allowing two parties to exchange a shared secret key, secure against passive adversaries.
Algorithm (encryption)	a set of mathematical rules (logic) used in the processes of encryption and decryption.
Algorithm (hash)	a set of mathematical rules (logic) used in the processes of message digest creation and key/signature generation.
Anonymity	of unknown or undeclared origin or authorship, concealing an entity's identification.
ANSI (American National Standards Institute)	develops standards through various Accredited Standards Committees (ASC). The X9 committee focuses on security standards for the financial services industry.
API (Application Programming Interface)	provides the means to take advantage of software features, allowing dissimilar software products to interact upon one another.
ASN.1 (Abstract Syntax Notation One)	ISO/IEC standard for encoding rules used in ANSI X.509 certificates, two types exist - DER (Distinguished Encoding Rules) and BER (Basic Encoding Rules).

Asymmetric keys	a separate but integrated user key-pair, comprised of one public key and one private key. Each key is one way, meaning that a key used to encrypt information can not be used to decrypt the same data.
Authentication	to prove genuine by corroboration of the identity of an entity.
Authorization certificate	an electronic document to prove one's access or privilege rights, also to prove one is who they say they are.
Authorization	to convey official sanction, access or legal power to an entity.
Blind signature	ability to sign documents without knowledge of content, similar to a notary public.
Block cipher	a symmetric cipher operating on blocks of plain text and cipher text, usually 64 bits.
Blowfish	a 64-bit block symmetric cipher consisting of key expansion and data encryption. A fast, simple, and compact algorithm in the public domain written by Bruce Schneier.
CA (Certificate Authority)	a trusted third party (TTP) who creates certificates that consist of assertions on various attributes and binds them to an entity and/or to their public key.
CAPI (Crypto API)	Microsoft's crypto API for Windows-based operating systems and applications.
Capstone	an NSA-developed cryptographic chip that implements a US government Key Escrow capability.
CAST	a 64-bit block cipher using 64-bit key, six S-boxes with 8-bit input and 32-bit output, developed in Canada by Carlisle Adams and Stafford Tavares.
CBC (Cipher Block Chaining)	the process of having plain text XORed with the previous cipher text block before it is encrypted, thus adding a feedback mechanism to a block cipher.
CDK (Crypto Developer Kit)	a documented environment, including an API for third parties to write secure applications using a specific vendor's cryptographic library.
CERT (Computer Emergency Response Team)	security clearinghouse that promotes security awareness. CERT provides 24-hour technical assistance for computer and network security incidents. CERT is located at the Software Engineering Institute at Carnegie Mellon University in Pittsburgh, PA.

Certificate (digital certificate)	an electronic document attached to a public key by a trusted third party, which provides proof that the public key belongs to a legitimate owner and has not been compromised.
CFM (Cipher Feedback Mode)	a block cipher that has been implemented as a self-synchronizing stream cipher.
CDSA (Common Data Security Architecture)	Intel Architecture Labs (IAL) developed this framework to address the data security problems inherent to Internet and Intranet for use in Intel and others' Internet products.
Certification	endorsement of information by a trusted entity.
CHAP (Challenge Authentication Protocol)	a session-based, two-way password authentication scheme.
Cipher text	the result of manipulating either characters or bits via substitution, transposition, or both.
Clear text	characters in a human readable form or bits in a machine-readable form (also called *plain text*).
Confidentiality	the act of keeping something private and secret from all but those who are authorized to see it.
Cookie	Persistent Client State HTTP Cookie - a file or token of sorts, that is passed from the web server to the web client (your browser) that is used to identify you and could record personal information such as ID and password, mailing address, credit card number, and other information.
CRAB	a 1024-byte block cipher (similar to MD5), using techniques from a one-way hash function, developed by Burt Kaliski and Matt Robshaw at RSA Laboratories.
Credentials	something that provides a basis for credit or confidence.
CRL (Certificate Revocation List)	an online, up-to-date list of previously issued certificates that are no longer valid.
Cross-certification	two or more organizations or Certificate Authorities that share some level of trust.
Cryptanalysis	the art or science of transferring cipher text into plain text without initial knowledge of the key used to encrypt the plain text.
CRYPTOKI	same as PKCS #11.

Cryptography	the art and science of creating messages that have some combination of being private, signed, unmodified with non-repudiation.
Cryptosystem	a system comprised of cryptographic algorithms, all possible plain text, cipher text, and keys.
Data integrity	a method of ensuring information has not been altered by unauthorized or unknown means.
Decryption	the process of turning cipher text back into plain text.
DES (Data Encryption Standard)	a 64-bit block cipher, symmetric algorithm also known as Data Encryption Algorithm (DEA) by ANSI and DEA-1 by ISO. Widely used for over 20 years, adopted in 1976 as FIPS 46.
Dictionary attack	a calculated brute force attack to reveal a password by trying obvious and logical combinations of words.
Diffie-Hellman	the first public key algorithm, invented in 1976, using discrete logarithms in a finite field.
Digital cash	electronic money that stored and transferred through a variety of complex protocols.
Direct trust	an establishment of peer-to-peer confidence.
Discrete logarithm	the underlying mathematical problem used in/by asymmetric algorithms, like Diffie-Hellman and Elliptic Curve. It is the inverse problem of modular exponentiation, which is a one-way function.
DMS (Defense Messaging System)	standards designed by the U.S. Department of Defense to provide a secure and reliable enterprise-wide messaging infrastructure for government and military agencies.
DNSSEC (Domain Name System Security Working Group)	a proposed *IETF* draft that will specify enhancements to the DNS protocol to protect the DNS against unauthorized modification of data and against masquerading of data origin. It will add data integrity and authentication capabilities to the DNS via digital signatures.
DSA (Digital Signature Algorithm)	a public key digital signature algorithm proposed by NIST for use in DSS.
Digital signature	an electronic identification of a person or thing created by using a public key algorithm. Intended to verify to a recipient the integrity of data and identity of the sender of the data.

DSS (Digital Signature Standard)	a NIST proposed standard (FIPS) for digital signatures using DSA.
ECC (Elliptic Curve Cryptosystem)	a unique method for creating public key algorithms based on mathematical curves over finite fields or with large prime numbers.
EDI (Electronic Data Interchange)	the direct, standardized computer-to-computer exchange of business documents (purchase orders, invoices, payments, inventory analyses, and others) between your organization and your suppliers and customers.
EES (Escrowed Encryption Standard)	a proposed U.S. government standard for escrowing private keys.
El Gamal scheme	used for both digital signatures and encryption based on discrete logarithms in a finite field; can be used with the DSA function.
Encryption	the process of disguising a message in such a way as to hide its substance.
Entropy	a mathematical measurement of the amount of uncertainty or randomness.
FEAL	a block cipher using 64-bit block and 64-bit key, design by A.Shimizu and S.Miyaguchi at NTT Japan.
Filter	a function, set of functions, or combination of functions that applies some number of transforms to its input set, yielding an output set containing only those members of the input set that satisfy the transform criteria. The selected members may or may not be further transformed in the resultant output set. An example would be a search function that accepts multiple strings having a boolean relationship `((` `like` *`a`* `or like` *`b`* `) but not containing` *`c`*`)`, and optionally forces the case of the found strings in the resultant output.
Fingerprint	a unique identifier for a key that is obtained by hashing specific portions of the key data.
FIPS (Federal Information Processing Standard)	a U.S. government standard published by NIST.
Firewall	a combination of hardware and software that protects the perimeter of the public/private network against certain attacks to ensure some degree of security.

GAK (Government Access to Keys)	a method for the government to escrow individual's private key.
Gost	a 64-bit symmetric block cipher using a 256-bit key, developed in the former Soviet Union.
GSS-API (Generic Security Services API)	a high-level security API based upon IETF RFC 1508, which isolates session-oriented application code from implementation details.
Hash function	a one-way hash function - a function that produces a message digest that cannot be reversed to produced the original.
HMAC	a key-dependent one-way hash function specifically intended for use with MAC (Message Authentication Code), and based upon IETF RFC 2104.
Hierarchical trust	a graded series of entities that distribute trust in an organized fashion, commonly used in ANSI X.509 issuing certifying authorities.
HTTP (HyperText Transfer Protocol)	a common protocol used to transfer documents between servers or from a server to a client.
IDEA (International Data Encryption Standard)	a 64-bit block symmetric cipher using 128-bit keys based on mixing operations from different algebraic groups. Considered one of the strongest algorithms.
IETF (Internet Engineering Task Force)	a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interested individual.
Identity certificate	a signed statement that binds a key to the name of an individual and has the intended meaning of delegating authority from that named individual to the public key.
Initialization vector (IV)	a block of arbitrary data that serves as the starting point for a block cipher using a chaining feedback mode (see cipher block chaining).
Integrity	assurance that data is not modified (by unauthorized persons) during storage or transmittal.
IPSec	a TCP/IP layer encryption scheme under consideration within the IETF.

ISA/KMP (Internet Security Association, Key Mgt. Protocol)	defines the procedures for authenticating a communicating peer, creation and management of Security Associations, key generation techniques, and threat mitigation, for example, denial of service and replay attacks.
ISO (International Organization for Standardization)	responsible for a wide range of standards, like the OSI model and international relationship with ANSI on X.509.
ITU-T (International Telecommunication Union-Telecommunication)	formally the CCITT (Consultative Committee for International Telegraph and Telephone), a worldwide telecommunications technology standards organization.
Kerberos	a trusted-third-party authentication protocol developed at MIT.
Key	a means of gaining or preventing access, possession, or control represented by any one of a large number of values.
Key escrow/recovery	a mechanism that allows a third party to retrieve the cryptographic keys used for data confidentiality, with the ultimate goal of recovery of encrypted data.
Key exchange	a scheme for two or more nodes to transfer a secret session key across an unsecured channel.
Key length	the number of bits representing the key size; the longer the key, the stronger it is.
Key management	the process and procedure for safely storing and distributing accurate cryptographic keys, the overall process of generating and distributing cryptographic key to authorized recipients in a secure manner.
Key splitting	a process for dividing portions of a single key between multiple parties, none having the ability to reconstruct the whole key.
LDAP (Lightweight Directory Access Protocol)	a simple protocol that supports access and search operations on directories containing information such as names, phone numbers, and addresses across otherwise incompatible systems over the Internet.
Lexical section	a distinct portion of a message that contains a specific class of data, for example, clear-signed data, encrypted data, and key data.
MAA (Message Authenticator Algorithm)	an ISO standard that produces a 32-bit hash, designed for IBM mainframes.

MAC (Message Authentication Code)	a key-dependent one-way hash function, requiring the use of the identical key to verify the hash.
MD2 (Message Digest 2)	128-bit one-way hash function designed by Ron Rivest, dependent on a random permutation of bytes.
MD4 (Message Digest 4)	128-bit one-way hash function designed by Ron Rivest, using a simple set of bit manipulations on 32-bit operands.
MD5 (Message Digest 5)	improved, more complex version of MD4, but still a 128-bit one-way hash function.
Message digest	a number that is derived from a message. Change a single character in the message and the message will have a different message digest.
MIC (Message Integrity Check)	originally defined in PEM for authentication using MD2 or MD5. Micalg (message integrity calculation) is used in secure MIME implementations.
MIME (Multipurpose Internet Mail Extensions)	a freely available set of specifications that offers a way to interchange text in languages with different character sets, and multi-media e-mail among many different computer systems that use Internet mail standards.
MMB (Modular Multiplication-based Block)	based on IDEA, Joan Daemen developed this 128-bit key /128-bit block size symmetric algorithm, not used because of its susceptibility to linear cryptanalysis.
MOSS (MIME Object Security Service)	defined in RFC 1848, it facilitates encryption and signature services for MIME, including key management based on asymmetric techniques (not widely used).
MSP (Message Security Protocol)	the military equivalent of PEM, an X.400-compatible application level protocol for securing e-mail, developed by the NSA in late 1980.
MTI	a one-pass key agreement protocol by Matsumoto, Takashima, and Imai that provides mutual key authentication without key confirmation or entity authentication.
NAT (Network Address Translator)	RFC 1631, a router connecting two networks together; one designated as inside, is addressed with either private or obsolete addresses that need to be converted into legal addresses before packets are forwarded onto the other network (designated as outside).
NIST (National Institute for Standards and Technology)	a division of the U.S. Dept. of Commerce that publishes open, interoperability standards called FIPS.

Non-repudiation	preventing the denial of previous commitments or actions.
Oakely	the "Oakley Session Key Exchange" provides a hybrid Diffie-Hellman session key exchange for use within the ISA/KMP framework. Oakley provides the important property of "Perfect Forward Secrecy."
One-time pad	a large non-repeating set of truly random key letters used for encryption, considered the only perfect encryption scheme, invented by Major J. Mauborgne and G. Vernam in 1917.
One-way hash	a function of a variable string to create a fixed length value representing the original pre-image, also called message digest, fingerprint, message integrity check (MIC).
Orange Book	the National Computer Security Center book entitled *Department of Defense Trusted Computer Systems Evaluation Criteria* that defines security requirements.
PAP (Password Authentication Protocol)	an authentication protocol that allows PPP peers to authenticate one another, does not prevent unauthorized access but merely identifies the remote end.
Passphrase	an easy-to-remember phrase used for better security than a single password; key crunching converts it into a random key.
Password	a sequence of characters or a word that a subject submits to a system for purposes of authentication, validation, or verification.
PCT (Private Communication Technology)	a protocol developed by Microsoft and Visa for secure communications on the Internet.
PEM (Privacy Enhanced Mail)	a protocol to provide secure internet mail, (RFC 1421-1424) including services for encryption, authentication, message integrity, and key management. PEM uses ANSI X.509 certificates.
Perfect forward secrecy	a cryptosystem in which the cipher text yields no possible information about the plain text, except possibly the length.
Primitive filter	a function that applies a single transform to its input set, yielding an output set containing only those members of the input set that satisfy the transform criteria. An example would be a search function that accepts only a single string and outputs a list of line numbers where the string was found.

Pretty Good Privacy (PGP)	an application and protocol (RFC 1991) for secure e-mail and file encryption developed by Phil R. Zimmermann. Originally published as Freeware, the source code has always been available for public scrutiny. PGP uses a variety of algorithms, like IDEA, RSA, DSA, MD5, SHA-1 for providing encryption, authentication, message integrity, and key management. PGP is based on the "Web-of-Trust" model and has worldwide deployment.
PGP/MIME	an IETF standard (RFC 2015) that provides privacy and authentication using the Multipurpose Internet Mail Extensions (MIME) security content types described in RFC1847, currently deployed in PGP 5.0 and later versions.
PKCS (Public Key Crypto Standards)	a set of *de facto* standards for public key cryptography developed in cooperation with an informal consortium (Apple, DEC, Lotus, Microsoft, MIT, RSA, and Sun) that includes algorithm-specific and algorithm-independent implementation standards. Specifications defining message syntax and other protocols controlled by RSA Data Security Inc.
PKI (Public Key Infrastructure)	a widely available and accessible certificate system for obtaining an entity's public key with some degree of certainty that you have the "right" key and that it has not been revoked.
Plain text (or clear text)	the human readable data or message before it is encrypted.
Pseudo-random number	a number that results from applying randomizing algorithms to input derived from the computing environment, for example, mouse coordinates. See *random number*.
Private key	the privately held "secret" component of an integrated asymmetric key pair, often referred to as the decryption key.
Public key	the publicly available component of an integrated asymmetric key pair often referred to as the encryption key.
RADIUS (Remote Authentication Dial-In User Service)	an IETF protocol (developed by Livingston, Enterprise), for distributed security that secures remote access to networks and network services against unauthorized access. RADIUS consists of two pieces - authentication server code and client protocols.
Random number	an important aspect to many cryptosystems, and a necessary element in generating a unique key(s) that are unpredictable to an adversary. True random numbers are usually derived from analog sources, and usually involve the use of special hardware.

RC2 (Rivest Cipher 2)	variable key size, 64-bit block symmetric cipher, a trade secret held by RSA, SDI.
RC4 (Rivest Cipher 4)	variable key size stream cipher, once a proprietary algorithm of RSA Data Security, Inc.
RC5 (Rivest Cipher 5)	a block cipher with a variety of arguments, block size, key size, and number of rounds.
RIPE-MD	an algorithm developed for the European Community's RIPE project, designed to resist known cryptanalysis attacks and produce a 128-bit hash value, a variation of MD4.
REDOC	a U.S.-patented block cipher algorithm developed by M. Wood, using a 160-bit key and an 80-bit block.
Revocation	retraction of certification or authorization.
RFC (Request for Comment)	an IETF document, either FYI (For Your Information) RFC sub-series that are overviews and introductory or STD RFC sub-series that identify specify Internet standards. Each RFC has an RFC number by which it is indexed and by which it can be retrieved (www.ietf.org).
ROT-13 (Rotation Cipher)	a simple substitution (Caesar) cipher, rotating each 26 letters 13 places.
RSA	short for RSA Data Security, Inc.; or referring to the principals - Ron Rivest, Adi Shamir, and Len Adleman; or referring to the algorithm they invented. The RSA algorithm is used in public key cryptography and is based on the fact that it is easy to multiply two large prime numbers together, but hard to factor them out of the product.
SAFER (Secure And Fast Encryption Routine)	a non-proprietary block cipher 64-bit key encryption algorithm. It is not patented, is available license free, and was developed by Massey, who also developed IDEA.
Salt	a random string that is concatenated with passwords (or random numbers) before being operated on by a one-way function. This concatenation effectively lengthens and obscures the password, making the cipher text less susceptible to dictionary attacks.
SDSI (Simple Distributed Security Infrastructure)	a new *PKI* proposal from Ronald L. Rivest (MIT), and Butler Lampson (Microsoft). It provides a means of defining groups and issuing group-membership, access-control lists, and security policies. SDSI's design emphasizes linked local name spaces rather than a hierarchical global name space.

SEAL (Software-optimized Encryption ALgorithm)	a fast stream cipher for 32-bit machines designed by Rogaway and Coppersmith.
Secret key	either the “private key” in public key (asymmetric) algorithms or the “session key” in symmetric algorithms.
Secure channel	a means of conveying information from one entity to another such that an adversary does not have the ability to reorder, delete, insert, or read (SSL, IPSec, whispering in someone’s ear).
Self-signed key	a public key that has been signed by the corresponding private key for proof of ownership.
SEPP (Secure Electronic Payment Protocol)	an open specification for secure bankcard transactions over the Internet. Developed by IBM, Netscape, GTE, Cybercash, and MasterCard.
SESAME (Secure European System for Applications in a Multi-vendor Environment)	European research and development project that extended Kerbros by adding authorization and access services.
Session key	the secret (symmetric) key used to encrypt each set of data on a transaction basis. A different session key is used for each communication session.
SET (Secure Electronic Transaction)	provides for secure exchange of credit card numbers over the Internet.
SHA-1 (Secure Hash Algorithm)	the 1994 revision to SHA, developed by NIST, (FIPS 180-1) used with DSS produces a 160-bit hash, similar to MD4, which is very popular and is widely implemented.
Single sign-on	one log-on provides access to all resources of the network.
SKIP (Simple Key for IP)	simple key-management for Internet protocols, developed by Sun Microsystems, Inc.
Skipjack	the 80-bit key encryption algorithm contained in NSA’s Clipper chip.
SKMP (Secure key Management Protocol)	an IBM proposed key-recovery architecture that uses a key encapsulation technique to provide the key and message recovery to a trusted third-party escrow agent.

S/MIME (Secure Multipurpose Mail Extension)	a proposed standard developed by Deming software and RSA Data Security for encrypting and/or authenticating MIME data. S/MIME defines a format for the MIME data, the algorithms that must be used for interoperability (RSA, RC2, SHA-1), and the additional operational concerns such as ANSI X.509 certificates and transport over the Internet.
SNAPI (Secure Network API)	a Netscape driven API for security services that provide ways for resources to be protected against unauthorized users, for communication to be encrypted and authenticated, and for the integrity of information to be verified.
SPKI (Simple Public Key Infrastructure)	an IETF proposed draft standard, (by Ellison, Frantz, and Thomas) public key certificate format, associated signature and other formats, and key acquisition protocol. Recently merged with Ron Rivest’s SDSI proposal.
SSH (Secure Shell)	an IETF proposed protocol for securing the transport layer by providing encryption, cryptographic host authentication, and integrity protection.
SSH (Site Security Handbook)	the Working Group (WG) of the Internet Engineering Task Force has been working since 1994 to produce a pair of documents designed to educate the Internet community in the area of security. The first document is a complete reworking of RFC 1244, and is targeted at system and network administrators, as well as decision makers (middle management).
SSL (Secure Socket Layer)	developed by Netscape to provide security and privacy over the Internet. Supports server and client authentication and maintains the security and integrity of the transmission channel. Operates at the transport layer and mimics the “sockets library,” allowing it to be application independent. Encrypts the entire communication channel and does not support digital signatures at the message level.
SST (Secure Transaction Technology)	a secure payment protocol developed by Microsoft and Visa as a companion to the PCT protocol.
Stream cipher	a class of symmetric key encryption where transformation can be changed for each symbol of plain text being encrypted, useful for equipment with little memory to buffer data.
STU-III (Secure Telephone Unit)	NSA designed telephone for secure voice and low-speed data communications for use by the U.S. Dept. of Defense and their contractors.
Substitution cipher	the characters of the plain text are substituted with other characters to form the cipher text.

S/WAN (Secure Wide Area Network)	RSA Data Security, Inc. driven specifications for implementing IPSec to ensure interoperability among firewall and TCP/IP products. S/WAN's goal is to use IPSec to allow companies to mix-and-match firewall and TCP/IP stack products to build Internet-based Virtual Private Networks (VPNs).
Symmetric algorithm	a.k.a., conventional, secret key, and single key algorithms; the encryption and decryption key are either the same or can be calculated from one another. Two sub-categories exist - Block and Stream.
TACACS+ (Terminal Access Controller Access Control System)	a protocol that provides remote access authentication, authorization, and related accounting and logging services, used by Cisco Systems.
Timestamping	recording the time of creation or existence of information.
TLS (Transport Layer Security)	an IETF draft, version 1 is based on the Secure Sockets Layer (SSL) version 3.0 protocol, and provides communications privacy over the Internet.
TLSP (Transport Layer Security Protocol)	ISO 10736, draft international standard.
Transposition cipher	the plain text remains the same but the order of the characters is transposed.
Triple DES	an encryption configuration in which the DES algorithm is used three times with three different keys.
Trust	a firm belief or confidence in the honesty, integrity, justice, and/or reliability of a person, company, or other entity.
TTP (Trust Third-Party)	a responsible party in which all participants involved agree upon in advance, to provide a service or function, such as certification, by binding a public key to an entity, time-stamping, or key-escrow.
UEPS (Universal Electronic Payment System)	a smart-card (secure debit card) -based banking application developed for South Africa where poor telephones make on-line verification impossible.
Validation	a means to provide timeliness of authorization to use or manipulate information or resources.
Verification	to authenticate, confirm, or establish accuracy.

VPN (Virtual Private Network)	allows private networks to span from the end-user, across a public network (Internet) directly to the Home Gateway of choice, such as your company's Intranet.
WAKE (Word Auto Key Encryption)	produces a stream of 32-bit words, which can be XORed with plain text stream to produce cipher text, invented by David Wheeler.
Web of Trust	a distributed trust model used by PGP to validate the ownership of a public key where the level of trust is cumulative based on the individual's knowledge of the "introducers."
W3C (World Wide Web Consortium)	an international industry consortium founded in 1994 to develop common protocols for the evolution of the World Wide Web.
XOR	exclusive-or operation; a mathematical way to represent differences.
X.509v3	an ITU-T digital certificate that is an internationally recognized electronic document used to prove identity and public key ownership over a communication network. It contains the issuer's name, the user's identifying information, and the issuer's digital signature, as well as other possible extensions in version 3.
X9.17	an ANSI specification that details the methodology for generating random and pseudo-random numbers.

Index

Symbols

A

B

C

D

E

F

G

H

I

R

S

T

U

V

W

X

Z