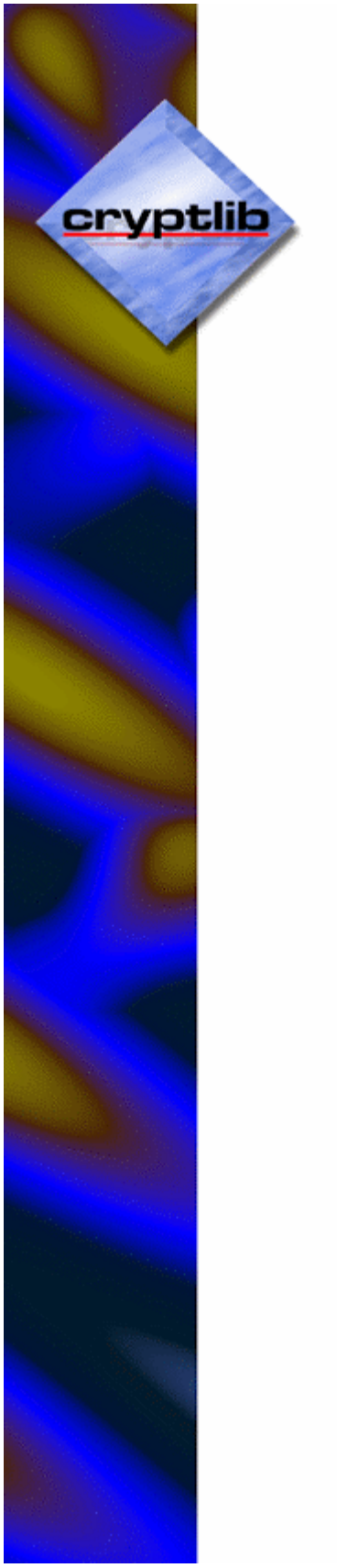


Web site: http://www.datasec.co.nz/, sales contact: sales@datasec.co.nz
Sales contact: Sean Rudd, phone +64 9 357-6323 x3313, fax +64 9 357-6324
Technical contact: Peter Gutmann, email: pgut001@cs.auckland.ac.nz

cryptlib

The Need for Security

The information age has seen the development of electronic pathways which carry vast amounts of valuable commercial, scientific, and educational information between financial institutions, companies, individuals, and government organisations. Unfortunately the unprecedented levels of access provided by systems like the Internet also expose this data to breaches of confidentiality, disruption of service, and outright theft. As a result, there is an enormous (and still growing) demand for the means to secure these online transactions. One report by the Computer Systems Policy Project (a consortium of virtually every large US computer company, including Apple, AT&T, Compaq, Digital, IBM, Silicon Graphics, Sun, and Unisys) estimates that the potential revenue arising from these security requirements in the US alone could be as much as US$30-60 billion by the year 2000, and the potential exposure to global users from a lack of this security is projected to reach between US$320 and 640 billion by the year 2000.

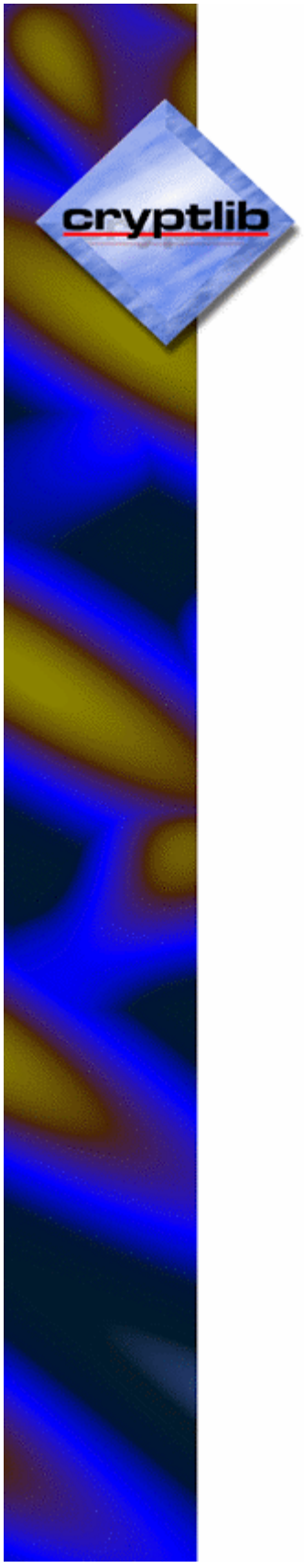
Unfortunately the security systems required to protect data are generally extremely difficult to design and implement, and even when available tend to require considerable understanding of the underlying principles in order to be used. This has lead to a proliferation of “snake oil” products which offer only illusionary security, or to organisations holding back from deploying online information systems because the means to secure them aren’t readily available, or (in the case of US products) because they employ weak, easily broken security which is unacceptable to users.

The cryptlib security toolkit is one answer to this problem. A complete description of the capabilities provided by cryptlib is given below.

cryptlib Overview

cryptlib is a powerful security toolkit which allows even inexperienced crypto programmers to easily add encryption and authentication security services to their software. The high-level interface provides anyone with the ability to add internationally recognised strong encryption and authentication capabilities to an application in as little as half an hour, without needing to know any of the low-level details which make the encryption or authentication work. Because of this, cryptlib dramatically reduces the cost involved in adding security to new or existing applications.

cryptlib provides a transparent and consistent interface to a number of widely-used security services and algorithms which are accessed through a straightforward, standardised interface with parameters such as the algorithm and key size being selectable by the user. Included as core components are implementations of the most popular encryption and authentication algorithms, Blowfish, CAST, DES, triple DES, IDEA, RC2, RC4, RC5, Safer, Safer-SK, and Skipjack, conventional encryption, MD2, MD4, MD5, RIPEMD-160 and SHA hash algorithms, HMAC-MD5, HMAC-SHA, HMAC-RIPEMD-160, and MDC-2 MAC algorithms, and Diffie-Hellman, DSA, Elgamal, and RSA public-key



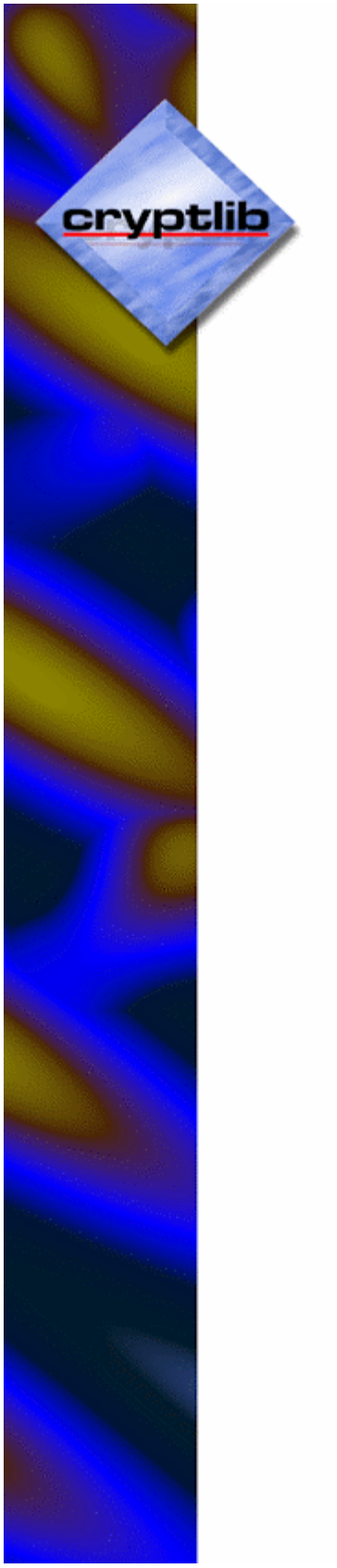
encryption, with elliptic-curve encryption currently under development. The algorithm parameters are summarised below:

Algorithm	Key size	Block size
Blowfish	448	64
CAST-128	128	64
DES	56	64
Triple DES	112 / 168	64
IDEA	128	64
RC2	1024	64
RC4	2048	8
RC5	832	64
Safer	128	64
Safer-SK	128	64
Skipjack	80	64
MD2	—	128
MD4	—	128
MD5	—	128
MDC-2	—	128
RIPEMD-160	—	160
SHA	—	160
HMAC-MD5	128	128
HMAC-SHA	160	160
HMAC-RIPEMD-160	160	160
Diffie-Hellman	4096	—
DSA	4096[1]	—
ElGamal	4096	—
RSA	4096	—

Unlike similar products sourced from the US, cryptlib contains no deliberately weakened encryption or backdoors, and allows worldwide use of keys of up to 4096 bits. In contrast products originating from the US contain either extremely weak encryption with keys a mere 40 bits in length (sometimes referred to as "8-cent keys" in reference to the cost of breaking one key), or, if they use longer keys, are required to contain backdoors which allow easy access by the US government (and, by extension, US business interests) to all data "protected" by the encryption. This makes US products unsuited for protecting sensitive, confidential data, and gives cryptlib an automatic advantage over all US products.

On top of the basic encryption services, cryptlib provides an extensive range of high-level capabilities including full X.509 certificate handling with support for all X.509v3 and IETF PKIX certificate features as well as support for SET, Microsoft AuthentiCode, S/MIME, and SSL client and server certificates, handling of certification requests and CRL's including automated checking of certificates against CRL's, creation and processing of PKCS #7 certificate chains, and a full range of certification authority (CA) functions. Alongside the certificate handling, cryptlib provides a sophisticated key database interface which allows the use of a wide range of key database types ranging from simple PGP keyrings

1 The DSA standard only defines key sizes from 512 to 1024 bits, cryptlib supports longer keys but there is no extra security to be gained from using these keys.



through to commercial-grade RDBMS's and LDAP directories with optional SSL protection. To complement its key management capabilities, cryptlib provides a complete S/MIME implementation with full-strength encryption, allowing email, files, and EDI transactions to be authenticated with digital signatures and encrypted in an industry-standard format.

In addition to its built-in capabilities, cryptlib can make use of the crypto capabilities of a variety of external crypto devices such as hardware crypto accelerators, Fortezza cards, PKCS #11 devices, and crypto smart cards. The crypto device interface also provides a convenient general-purpose plug-in capability for adding new functionality which will be automatically used by cryptlib.

cryptlib features

cryptlib provides a standardised interface to a number of popular encryption algorithms, as well as providing a high-level interface which hides most of the implementation details and provides an operating-system-independent encoding method which makes it easy to transfer secured data from one operating environment to another. Although use of the high-level interface is recommended, experienced programmers can directly access the lower-level encryption routines for implementing custom encryption protocols or methods not directly provided by cryptlib.

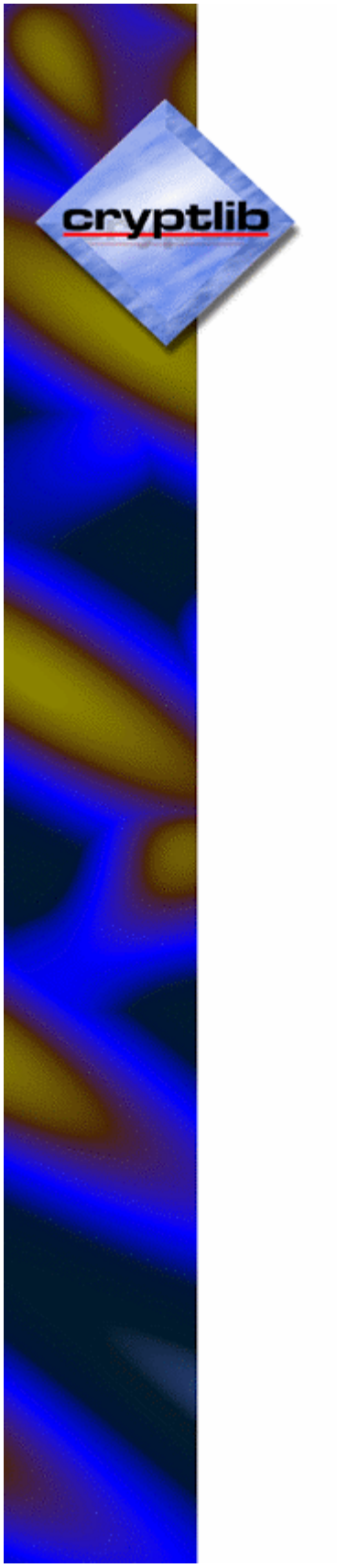
Programming Interface

The application programming interface (API) serves as an interface to a range of plug-in encryption modules which allow encryption algorithms to be added in a fairly transparent manner, so that adding a new algorithm or replacing an existing software implementation with custom encryption hardware can be done without any trouble. The standardised API allows any of the algorithms and modes supported by cryptlib to be used with a minimum of coding effort. In addition the easy-to-use high-level routines allow for the exchange of encrypted session keys and data and the creation and checking of digital signatures with a minimum of programming overhead.

cryptlib has been written to be as foolproof as possible. On initialization it performs extensive self-testing against test data from encryption standards documents, and the API's check each parameter and function call for errors before any actions are performed, with error reporting down to the level of individual parameters. In addition logical errors such as, for example, a key exchange function being called in the wrong sequence, are checked for and identified.

Standards Compliance

All algorithms, security methods, and data encoding systems in cryptlib either comply with one or more national and international banking and security standards, or are implemented and tested to conform to a reference implementation of a particular algorithm or security system. Compliance with national and international security standards is automatically provided when cryptlib is integrated into an application. These standards include ANSI X3.92, ANSI X3.106, ANSI X9.9, ANSI X9.17, ANSI X9.30-1, ANSI X9.30-2, ANSI



X9.31-1, FIPS PUB 46-2 FIPS PUB 74, FIPS PUB 81, FIPS PUB 113, FIPS PUB 180, FIPS PUB 180-1, FIPS PUB 186, ISO/IEC 8372, ISO/IEC 8731 ISO/IEC 8732, ISO/IEC 8824/ITU-T X.680, ISO/IEC 8825/ITU-T X.690, ISO/IEC 9797, ISO/IEC 10116, ISO/IEC 10118, PKCS #1, PKCS #3, PKCS #7, PKCS #9, PKCS #10, RFC 1319, RFC 1320, RFC 1321, RFC 1750, RFC 2104, RFC 2144, RFC 2268, RFC 2312, RFC 2313, RFC 2314, RFC 2315, and RFC 2459. Because of the use of internationally recognised and standardised security algorithms, cryptlib users will avoid the problems caused by homegrown, proprietary algorithms and security techniques which often fail to provide any protection against attackers, resulting in embarrassing bad publicity and expensive software recalls.

Y2K Compliance

cryptlib handles all date information using the ANSI/ISO C time format which does not suffer from Y2K problems. Although earlier versions of the X.509 certificate format do have Y2K problems, cryptlib transparently converts the date encoded in certificates to and from the ANSI/ISO format, so cryptlib users will never see this. cryptlib's own time/date format is not affected by any Y2K problems, and cryptlib itself conforms to the requirements in the British Standards Institutions DISC PD2000-1:1998 Y2K compliance standard.

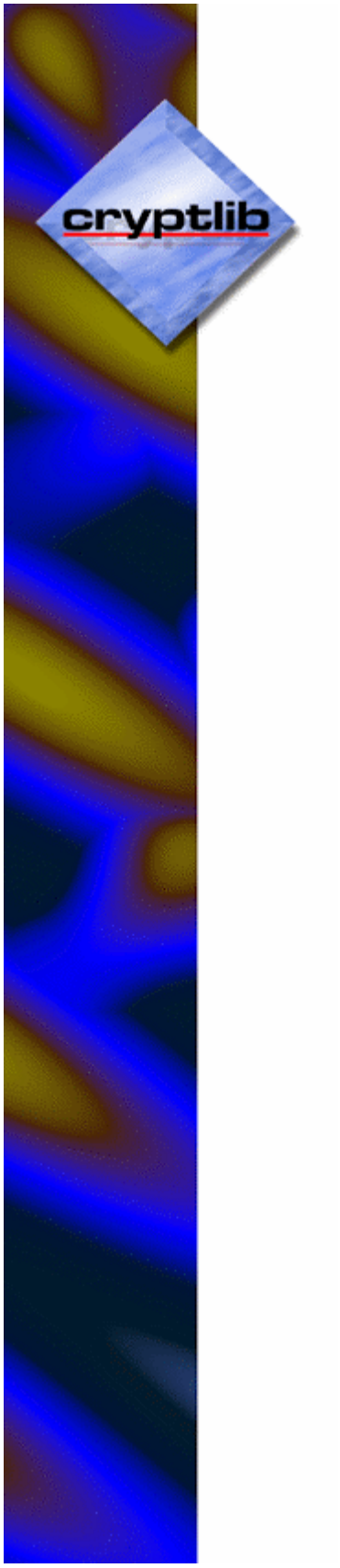
Encrypted Object Management

cryptlibs powerful object management interface provides the ability to add encryption and authentication capabilities to an application without needing to know all the low-level details which make the encryption or authentication work. The automatic object-management routines take care of encoding issues and cross-platform portability problems, so that a single function call is enough to export a public-key encrypted session key with all the associated information and parameters needed to recreate the session key on the other side of a communications channel, or to generate a digital signature on a piece of data. This provides a considerable advantage over other encryption toolkits which often require hundreds of lines of code and the manipulation of complex encryption data structures to perform the same task.

S/MIME

cryptlib employs the IETF-standardised Cryptographic Message Syntax (CMS, formerly called PKCS #7) format as its native data format. CMS is the underlying format used in the S/MIME secure mail standard, as well as a number of other standards covering secure EDI and related systems like HL7 messaging. As an example of its use in secure EDI, cryptlib provides security services for the Sypmhonia EDI messaging toolkit which is used to communicate medical lab reports, patient data, drug prescription information, and similar information requiring a high level of security.

The S/MIME implementation uses cryptlibs enveloping interface which allows simple, rapid integration of strong encryption and authentication capabilities into existing email agents and messaging software. The resulting signed data format provides message integrity and origin authentication services, the enveloped data format provides confidentiality. The complexity of the S/MIME format means



that the few other toolkits which are available require a high level of programmer knowledge of S/MIME processing issues. In contrast cryptlib's enveloping interface makes the process as simple as pushing raw data into an envelope and popping the processed data back out, a total of three function calls, plus one more call to add the appropriate encryption or signature key.

Certificate Management

cryptlib implements full X.509 certificate support, including all X.509 version 3 extensions as well as extensions defined in the IETF PKIX certificate profile. cryptlib also supports additional certificate types and extensions including SET certificates, Microsoft AuthentiCode and Netscape and Microsoft server-gated crypto certificates, S/MIME and SSL client and server certificates, and various vendor-specific extensions such as Netscape certificate types and the Thawte secure extranet.

In addition to certificate handling, cryptlib allows the generation of PKCS #10 certification requests with CMMF extensions suitable for submission to certification authorities (CA's) in order to obtain a certificate. Since cryptlib is itself capable of processing certification requests into certificates, it is also possible to use cryptlib to provide full CA services. cryptlib also supports the creating and handling of the certificate chains required for S/MIME, SSL, and other applications, and the creation of certificate revocation lists (CRL's) with the capability to check certificates against existing or new CRL's either automatically or under programmer control.

cryptlib can import and export certification requests, certificates, and CRL's in straight binary format, as PKCS #7 certificate chains, and as Netscape certificate sequences, with or without base64 armouring. This covers the majority of certificate and certificate transport formats used by a wide variety of software such as web browsers and servers.

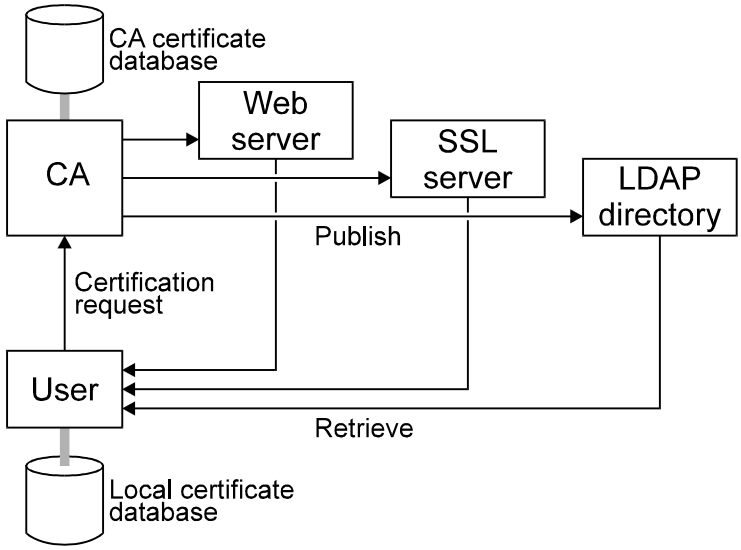
The certificate types which are supported include:

- Basic X.509 version 1 certificates
- Extended X.509 version 3 certificates
- Certificates conformant to the IETF PKIX profile
- SSL server and client certificates
- S/MIME email certificates
- SET certificiates
- AuthentiCode code signing certificates
- IPSEC server, client, end-user, and tunneling certificates
- Server-gated crypto certificates
- Timestamping certificates

In addition cryptlib supports all X.509v3, IETF, S/MIME, and SET certificate extensions and a many vendor-specific extensions including ones covering public and private key usage, certificate policies, path and name constraints, policy

constraints and mappings, and alternative names and other identifiers. This comprehensive coverage makes cryptlib a single solution for almost all certificate processing requirements.

The diagram below shows a typical cryptlib application, in which it provides the full functionality of both a CA (processing certification requests, storing the issued certificates locally in a certificate database, and optionally publishing the certificates on the web or in an LDAP directory) and an end entity (generating certification requests, submitting them to a CA, and retrieving the result from the web or a directory service).

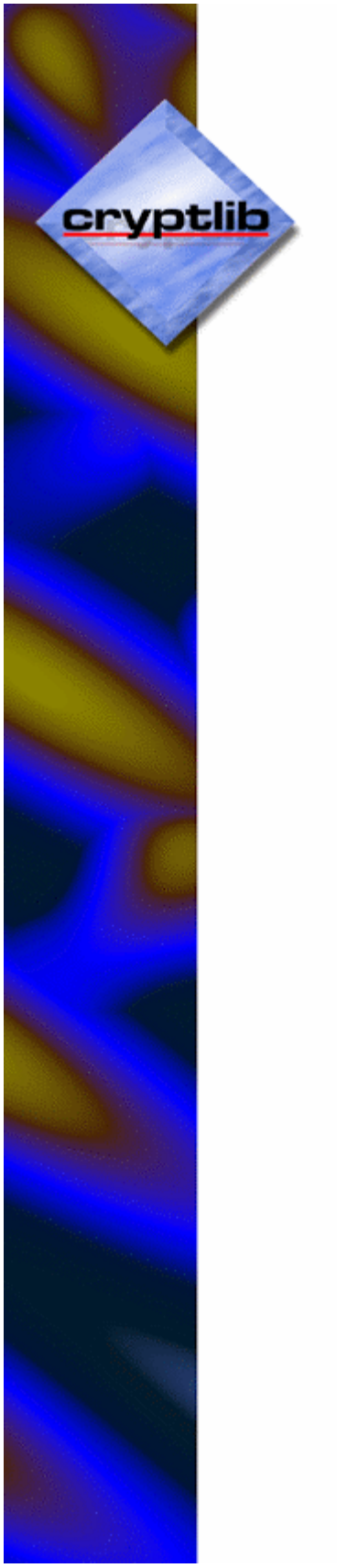


To handle certificate trust and revocation issues, cryptlib includes a certificate trust manager which can be used to automatically manage CA trust settings, for example a CA can be designated as a trusted issuer which will allow cryptlib to automatically evaluate trust along certificate chains. Similarly, cryptlib can automatically check certificates against CRL's published by CA's, removing from the user the need to perform complex manual checking.

Key Database Interface

cryptlib utilizes commercial-strength RDBMS's to store keys in the internationally standardised X.509 format. The key database integrates seamlessly into existing databases and can be managed using existing tools. For example a key database stored on an MS SQL Server might be managed using Visual Basic or MS Access; a key database stored on an Oracle server might be managed through SQL*Plus. cryptlib currently supports Beagle SQL, mSQL, MySQL, Oracle, Postgres, Raima Velocis, and Solid databases under Unix, and most databases which can be accessed through Windows ODBC drivers. This includes MS Access, dBase, Oracle, Paradox, MS SQL Server, and many more. Extending the interface to support new database types requires approximately 200 lines of code to tie the cryptlib routines into a particular database backend.

In addition to key databases, cryptlib supports the storage and retrieval of certificates in LDAP directories. This interface provides full LDAPv3 support, with optional SSL protection of the connection to the directory. cryptlib also



supports HTTP access for keys accessible via the web, as well as external flat-file key collections such as PGP key rings. The key collections may be freely mixed (so for example a private key could be stored in a disk file, a PGP keyring or on a smart card with the corresponding X.509 public key certificate being stored in an Oracle or SQL Server database, an LDAP directory, or on the web).

Private keys may be stored on disk encrypted with an algorithm such as triple DES (selectable by the user), with the password processed using several hundred iterations of a hashing algorithm such as SHA-1 (also selectable by the user). Where the operating system supports it, cryptlib will apply system security features such as ACL's under Windows NT and file permissions under Unix to the private key file to further restrict access.

Smart Card Support

cryptlib allows private keys to be stored on a variety of smart cards accessed through a selection of smart card readers — use of cryptlib won't tie you to a single card or reader vendor. As an extra precaution, cryptlib encrypts all data written to the smart card so that even if the card is hacked, the data remains secure. Support for new smart card types and/or readers can be added on request.

Crypto Devices

In addition to its built-in capabilities, cryptlib can make use of the crypto capabilities of a variety of external crypto devices such as:

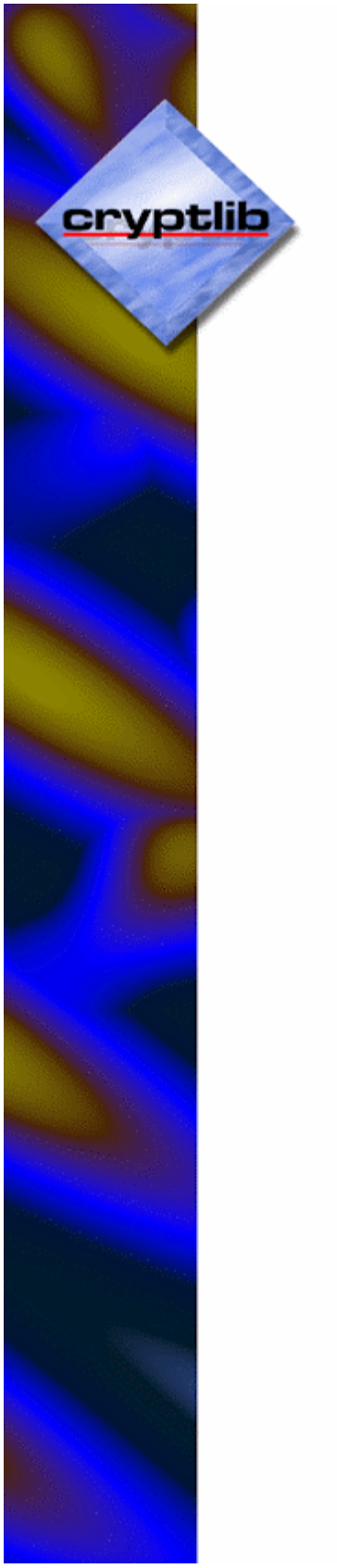
- Hardware crypto accelerators
- Fortezza cards
- PKCS #11 devices
- Crypto smart cards.

In addition, the crypto device interface provides a convenient general-purpose plug-in capability for adding new functionality which will be automatically used by cryptlib in its higher-level routines which handle key management, digital signatures, and message encryption.

Security Features

cryptlib implements a security perimeter around the encryption core, with all encryption related data being referred to through arbitrary handles which are used by cryptlib to reference data which is hidden from the calling program. No outside access to state variables or keying information is possible (unless the operating system security itself is compromised, and even then cryptlib takes steps to make outside access to the most sensitive information as difficult as possible). If the underlying implementation is in hardware, all cryptovariables will be securely locked inside the hardware with no external access possible.

If the operating system supports it, all sensitive information used will be page-locked to ensure it is never swapped to disk from where it could be recovered using a disk editor. All memory corresponding to security-related data is managed by cryptlib and will be automatically sanitized and freed when cryptlib shuts down



even if the calling program forgets to release the memory itself.

Where the operating system supports it, cryptlib will apply operating system security features to any objects it creates or manages. For example under Windows NT cryptlib private key files will be created with an access control list (ACL) which allows only the key owner access to the file; under Unix the file permissions will be set to achieve the same result.

Performance

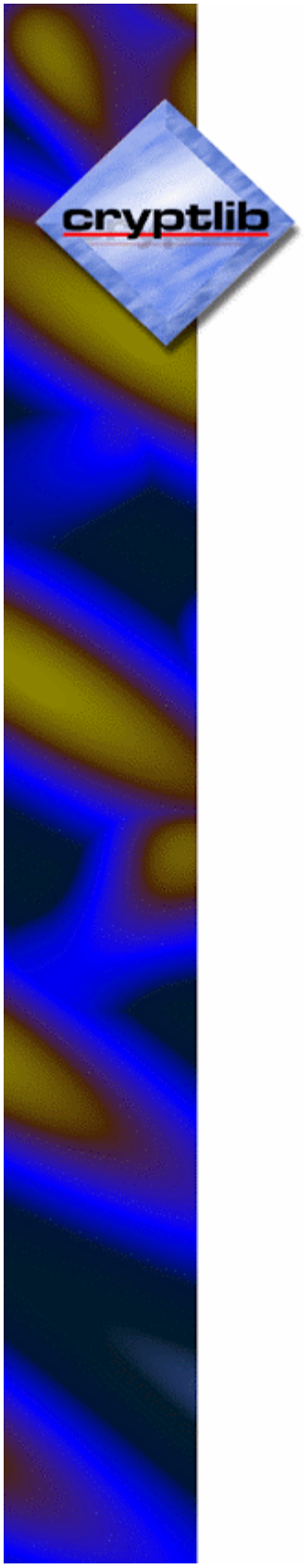
cryptlib is re-entrant and completely thread-safe, allowing it to be used with multithreaded applications under Windows 95/98 and Windows NT, OS/2, and Unix systems which support threads. Because it is thread-safe, lengthy cryptlib operations can be run in the background if required while other processing is performed in the foreground. In addition cryptlib itself is multithreaded so that computationally intensive internal operations take place in the background without impacting the performance of the calling application.

Most of the core algorithms used in cryptlib have been implemented in assembly language in order to provide the maximum possible performance. These routines provide an unprecedented level of performance, in some cases running faster than expensive, specialised encryption hardware designed to perform the same task. This means cryptlib can be used for high-bandwidth applications such as video/audio encryption and online network and disk encryption without the need to resort to expensive, hard-to-get encryption hardware.

Cryptographic Random Number Management

cryptlib contains an internal secure random data management system which provides the cryptographically strong random data used to generate session keys and public/private keys, in public-key encryption operations, and in various other areas which require secure random data. The random data pool is updated with unpredictable process-specific information as well as system-wide data such as current disk I/O and paging statistics, network, SMB, LAN manager, and NFS traffic, packet filter statistics, multiprocessor statistics, process information, users, VM statistics, process statistics, open files, inodes, terminals, vector processors, streams, and loaded code, objects in the global heap, loaded modules, running threads, process, and tasks, and an equally large number of system performance-related statistics covering virtually every aspect of the operation of the system.

The exact data collected depends on the hardware and operating system, but generally includes quite detailed operating statistics and information. In addition if a `/dev/random`-style randomness driver (which continually accumulates random data from the system) is available, cryptlib will use this is a source of randomness. Finally, cryptlib supports a number of cryptographically strong hardware random number generators such as the Protego SG100 and various serial-port-based generators which can be used to supplement or replace the internal generator. This level of secure random number management ensures that security problems such as those present in Netscape's web browser (which allowed encryption keys to be predicted without breaking the encryption because the random data gathered wasn't at all random) can't occur with cryptlib.



Web site: http://www.datasec.co.nz/, sales contact: sales@datasec.co.nz
Sales contact: Sean Rudd, phone +64 9 357-6323 x3313, fax +64 9 357-6324
Technical contact: Peter Gutmann, email: pgut001@cs.auckland.ac.nz

Configuration Options

cryptlib works with a configuration database which can be used to tune its operation for different environments using the Windows registry or Unix `rc` files. This allows a system administrator to set a consistent security policy (for example mandating the use of 1024-bit public keys on a company-wide basis instead of the insecure 512-bit keys used in most US-sourced products). These configuration options are then automatically applied by cryptlib to operations such as key generation and data encryption and signing, although they can be overridden on a per-application or per-user basis if required.

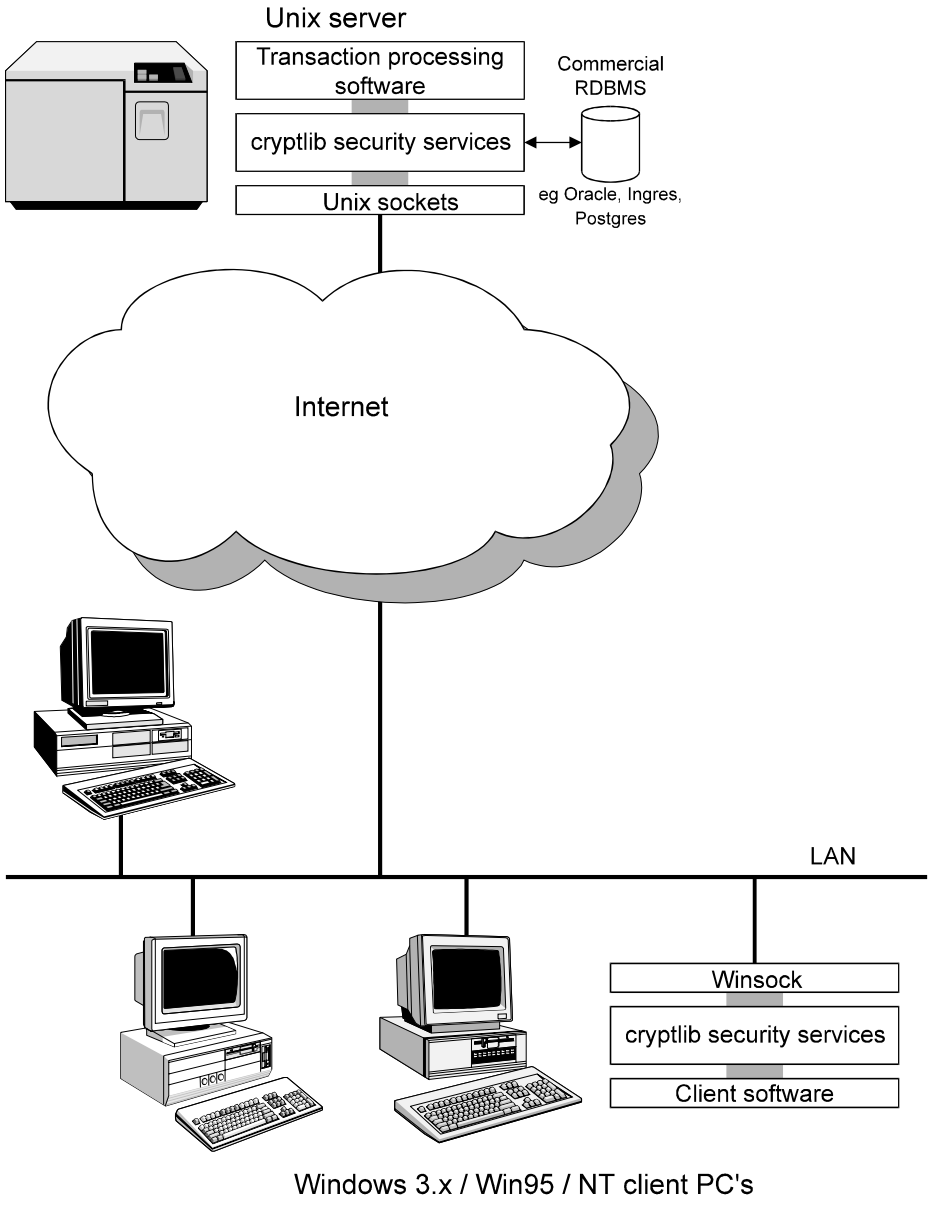
cryptlib Applications

The security services provided by cryptlib can be used in virtually any situation which requires the protection or authentication of sensitive data. Some areas in which cryptlib is currently used include:

- Protection of medical records transmitted over electronic links.
- Protection of financial information transmitted between branches of banks.
- Transparent disk encryption.
- Strong security services added to web browsers with weak, exportable security.
- Encrypted electronic mail.
- File encryption.
- Digitally signed electronic forms.
- Secure database access.

A typical application, in which a number of machines running various versions of Microsoft Windows use cryptlib to communicate securely with a server running Unix, is shown below:

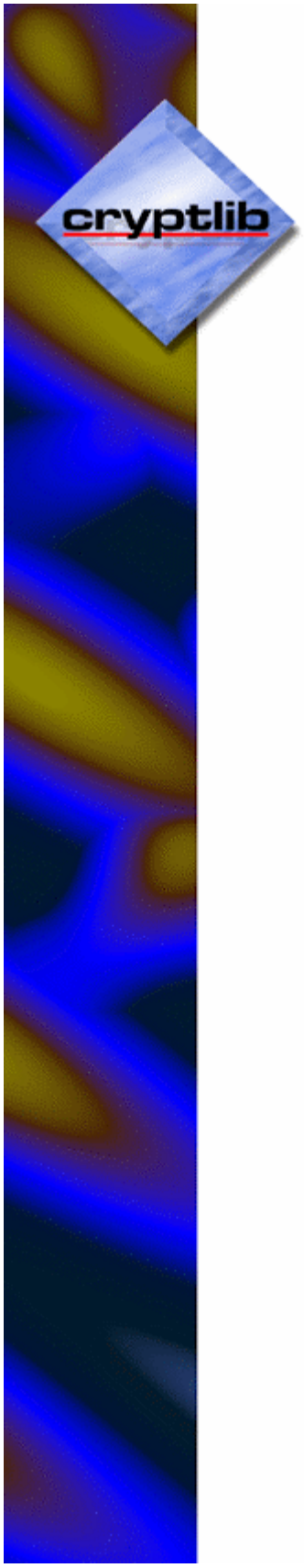
Web site: http://www.datasec.co.nz/, sales contact: sales@datasec.co.nz
Sales contact: Sean Rudd, phone +64 9 357-6323 x3313, fax +64 9 357-6324
Technical contact: Peter Gutmann, email: pgut001@cs.auckland.ac.nz



cryptlib Architecture

cryptlib consists of a set of layered security services and associated programming interfaces which provide an integrated set of information and communications security capabilities. Much like the OSI networking reference model, cryptlib contains a series of layers which provide each level of abstraction, with higher layers building on the capabilities provided by the lower layers.

At the lowest level are basic components such as core encryption and authentication routines, which are usually implemented in software but may also be implemented in hardware for speed (due to the speed of the software components used in cryptlib, some of the software is actually faster than dedicated hardware).

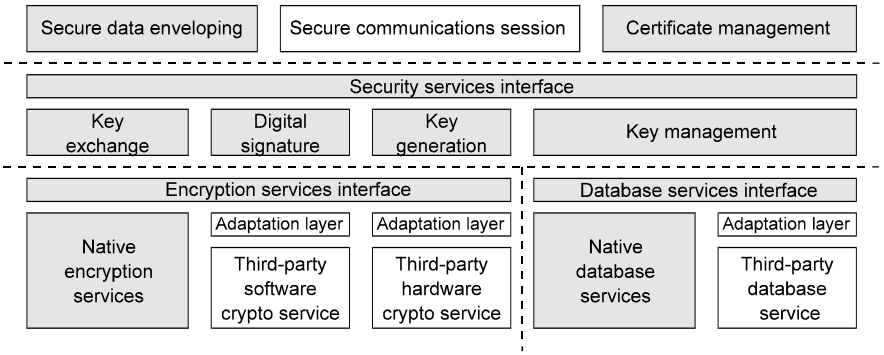


At the next level are components which wrap up the specialised and often quite complex core components in a layer which provides abstract functionality and ensures complete cross-platform portability of data. These functions typically cover areas such as “create a digital signature” or “exchange an encryption key”.

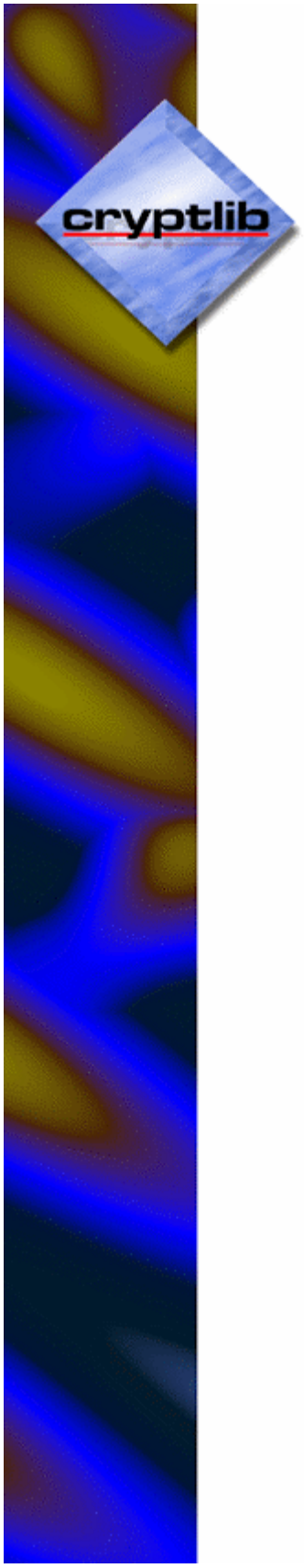
At the highest level are extremely powerful and easy-to-use functions such as “encrypt a message”, “sign a message”, and “create a digital certificate” which require no knowledge of encryption techniques, and which take care of complex issues such as key management, data encoding, en/decryption, and digital signature processing.

cryptlib consists of a multi-platform architecture which provides these services across all major operating system environments, including BeOS, DOS, OS/2, Windows 3.x, Windows 95/98, Windows NT, the Tandem environment, and a large variety of Unix versions such as AIX, A/UX, Digital Unix, DGUX, FreeBSD/NetBSD/OpenBSD, HPUX, IRIX, Linux, OSF/1, SCO, Solaris, SunOS, and Ultrix.

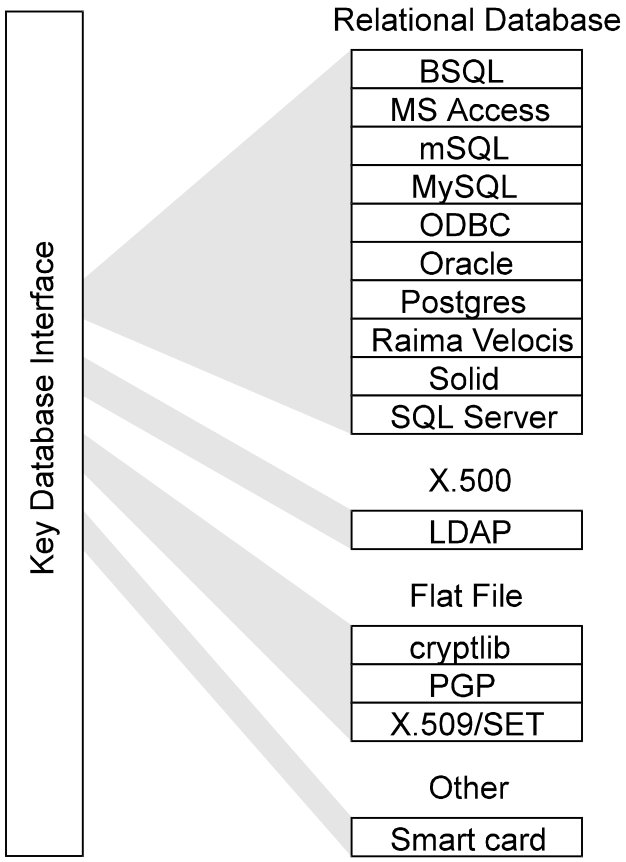
The overall cryptlib architecture is as follows:



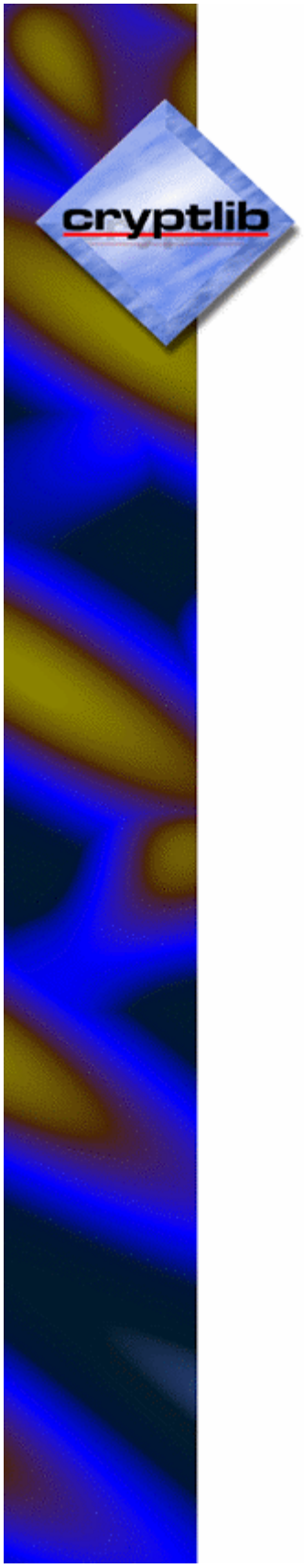
The key database services are implemented as an interface layer which communicates with various key storage technologies and mechanisms. This layer is as follows:



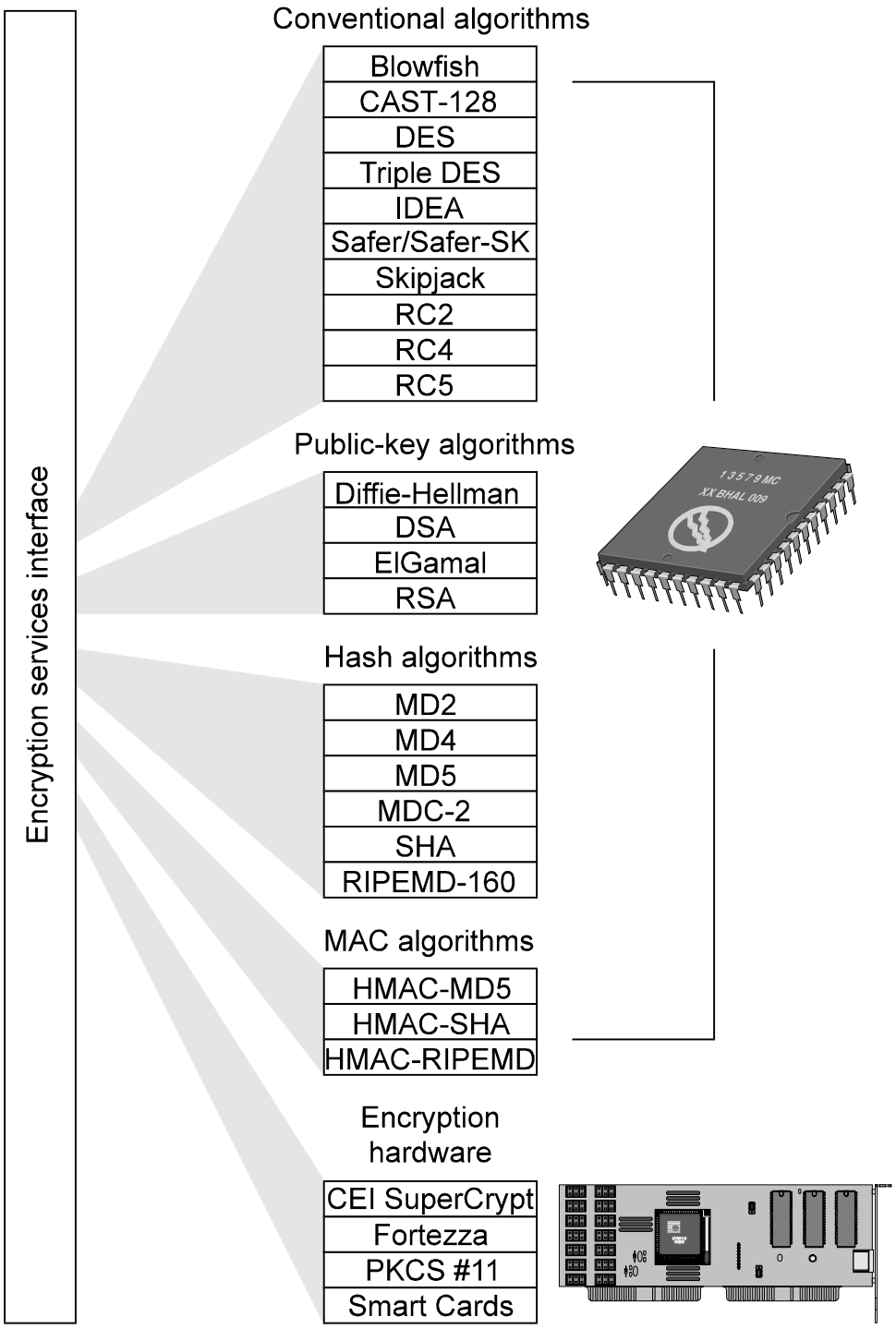
Web site: http://www.datasec.co.nz/, sales contact: sales@datasec.co.nz
Sales contact: Sean Rudd, phone +64 9 357-6323 x3313, fax +64 9 357-6324
Technical contact: Peter Gutmann, email: pgut001@cs.auckland.ac.nz



The low-level encryption services are implemented as plug-in modules which can be added and removed as required. This layer is as follows:

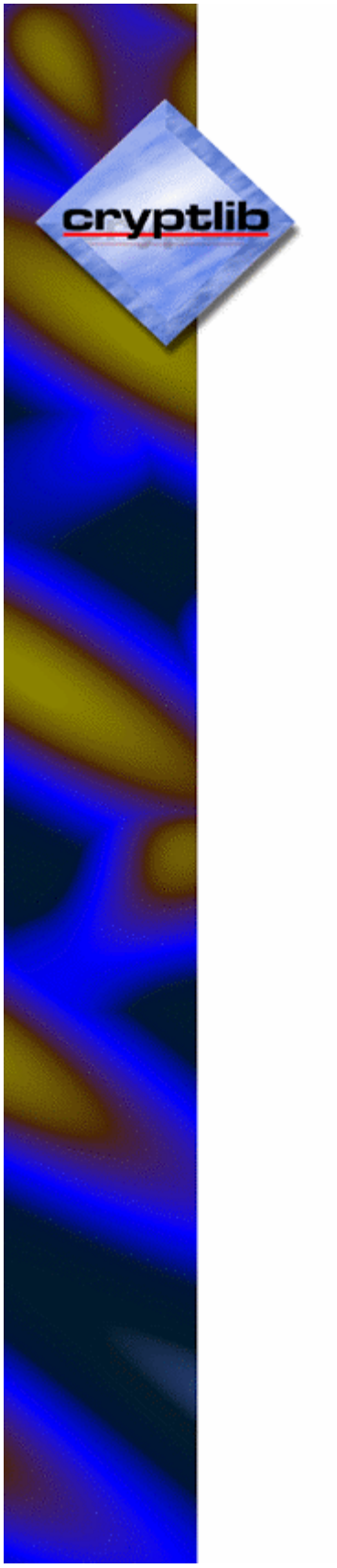


Web site: http://www.datasec.co.nz/, sales contact: sales@datasec.co.nz
Sales contact: Sean Rudd, phone +64 9 357-6323 x3313, fax +64 9 357-6324
Technical contact: Peter Gutmann, email: pgut001@cs.auckland.ac.nz



Encryption Code Example

The best way to illustrate what cryptlib can do is with an example. The following



code digitally signs a message:

```
/* Create an envelope for the message */
cryptCreateEnvelope( &cryptEnvelope );

/* Get our signature key from a smart card */
cryptKeysetOpen( &cryptKeyset, CRYPT_KEYSET_SMARTCARD, "Gemplus",
   CRYPT_KEYSET_READONLY );
cryptGetPrivateKey( cryptKeyset, &signatureKey, "John Doe",
   "Password" );
cryptKeysetClose( cryptKeyset );

/* Push our signature key into the envelope */
cryptAddEnvinfoNumeric( cryptEnvelope, CRYPT_ENVINFO_SIGNATURE,
   signatureKey );

/* Push in the message data and pop out the signed result */
cryptPushData( cryptEnvelope, message, messageSize, &bytesIn );
cryptPopData( cryptEnvelope, encryptedMessage, encryptedSize,
   &bytesOut );

/* Clean up */
cryptDestroyEnvelope( cryptEnvelope );
```

This performs the same task as a program like PGP, using just 6 function calls (and with smart card support which PGP doesn't have). All data management is handled automatically by cryptlib, so there's no need to worry about encryption modes and algorithms and keylengths and key types and initialisation vectors and other details (although cryptlib provides the ability to specify all this if you feel the need).

The code shown above results in cryptlib performing the following actions:

1. Hash the message using the default hash algorithm (usually SHA-1).
2. Sign the hash using the given signature key.
3. Wrap up the message in an OS-independent manner to allow it to be decoded on any platform.
4. Wrap up the signature alongside the signed message.
5. Pass the result back to the user.

However unless you want to call cryptlib using the low-level interface, you never need to know about any of this. cryptlib will automatically know what to do with the data based on the resources you add to the envelope — if you add a signature key it will sign the data, if you add an encryption key it will encrypt the data, and so on.

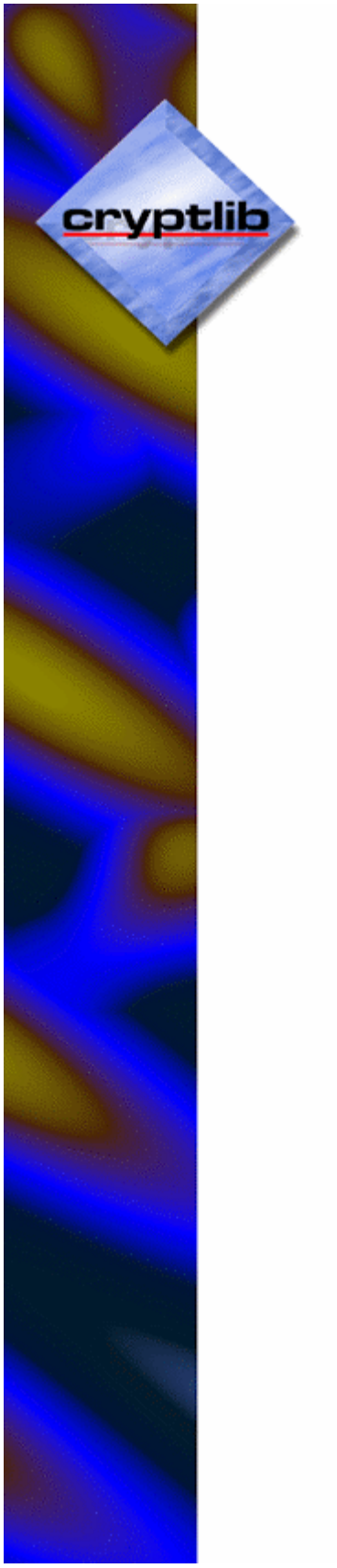
Certificate Management Code Example

The following code demonstrates cryptlibs certificate management capabilities by generating a certification request:

```
/* Create a certification request and add the public key to it */
cryptCreateCert( cryptCertRequest, CRYPT_CERTTYPE_CERTREQUEST );
cryptAddCertComponentNumeric( cryptCertRequest,
   CRYPT_CERTINFO_SUBJECTPUBLICKEYINFO, pubKeyContext );

/* Add identification information */
/* ... */

/* Sign the certification request with the private key and export it
```




```
   */
cryptSignCert( cryptCertRequest, privKeyContext );
cryptExportCert( certRequest, CRYPT_CERTFORMAT_CERTIFICATE,
   &certRequestLength, cryptCertRequest );

/* Destroy the certification request */
cryptDestroyCert( certRequest );
```

and converting it into an X.509v3 certificate by signing it with a key belonging to a certification authority (CA):

```
/* Import the certification request and check its validity */
cryptImportCert( certRequest, &cryptCertRequest );
cryptCheckCert( cryptCertRequest, CRYPT_UNUSED );

/* Create a certificate and add the information from the
   certification request to it */
cryptCreateCert( &cryptCert, CRYPT_CERTTYPE_CERTIFICATE );
cryptAddCertComponentNumeric( cryptCert, CRYPT_CERTINFO_CERTIFICATE,
   cryptCertRequest );

/* Sign the certificate with the CA's private key and export it */
cryptSignCert( cryptCert, caPrivateKey );
cryptExportCert( cert, CRYPT_CERTFORMAT_CERTIFICATE, &certLength,
   cryptCert );

/* Destroy the certificate and certification request */
cryptDestroyCert( cryptCert );
cryptDestroyCert( cryptCertRequest );
```

This code sample forms the basic core of a certification authority.

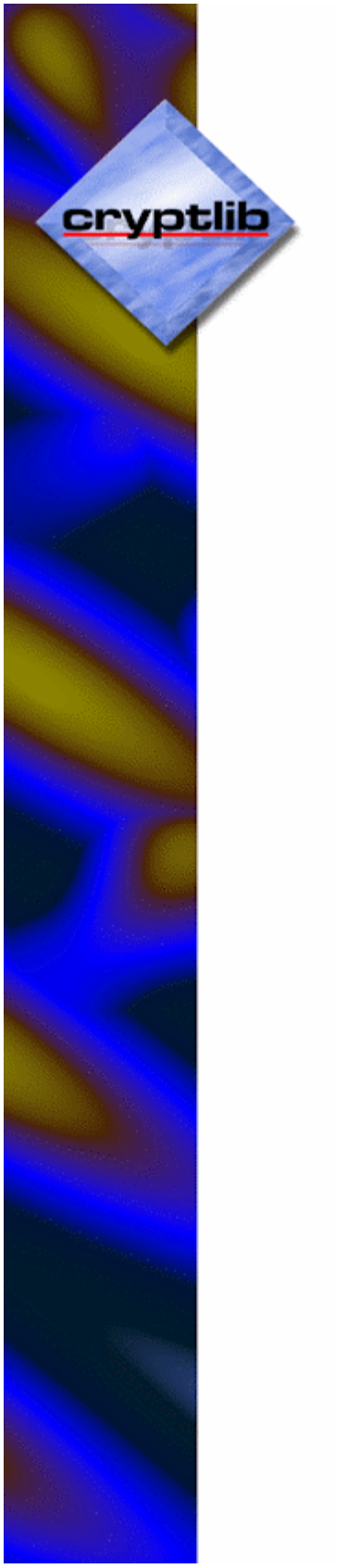
As with the encryption example, cryptlib is performing a great deal of work in the background, including automatic key management, encoding and decoding of certificate data, adding and processing X.509v3 certificate extensions, and performing signature and validity checking on the data.

Security Design and Consulting Services

In addition to providing the cryptlib components required to add encryption and security services to a particular application, we can also provide security design and consulting to assist customers in integrating cryptlib's security services into their projects. This allows us to provide a finished security solution for applications which include:

- Protection of medical records transmitted over electronic links.
- Encrypted electronic mail.
- Protection of financial information transmitted between branches of banks.
- Software authentication and secure software distribution.
- File encryption.
- Digitally signed electronic forms.
- Secure database access.
- Secure data/file backup facilities.

Our input can range from providing advice on the use of cryptlib through to creating a complete, customer-specific solution based on cryptlib components. If you would like to use our design and consulting services, please contact one of the



Web site: http://www.datasec.co.nz/, sales contact: sales@datasec.co.nz
Sales contact: Sean Rudd, phone +64 9 357-6323 x3313, fax +64 9 357-6324
Technical contact: Peter Gutmann, email: pgut001@cs.auckland.ac.nz

cryptlib salespeople with your requirements.

Pricing

Most other encryption libraries and toolkits are provided in an all-in-one form which bundles all the capabilities of the toolkit into a single package which must be licensed, at significant cost, as a single unit. In recognition of the fact that many users won't require some of the more advanced (and expensive) capabilities provided by cryptlib, it can be used as one of four versions:

Version	Features
cryptlib lite	Conventional encryption, hashing, and MAC algorithms and related functions to handle passwords and keys.
cryptlib cert	As cryptlib lite but with digital signatures, X.509 certificate management and key database. This is currently in beta, please contact the cryptlib technical contact for usage details.
cryptlib pro	As cryptlib cert but with digital enveloping and S/MIME support.

Most applications which require basic security features such as password encryption or the protection of sensitive information like database records will be able to use the relatively inexpensive cryptlib lite, more demanding applications can use the digital signature and X.509 certificate management features provided in cryptlib cert, and complete crypto-enabled applications can use the full range of services provided in cryptlib pro. Licensing terms can be based on the number of end users, use on servers, or using a royalty-based system.

Per-user licenses

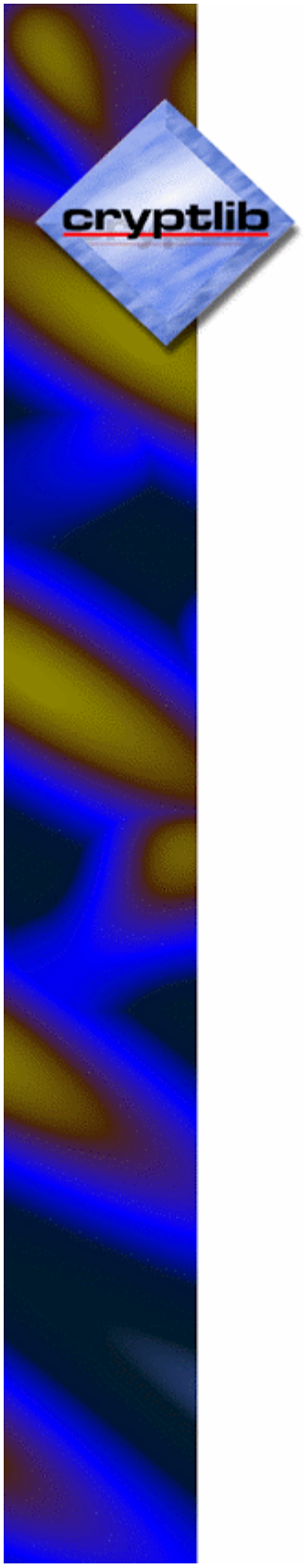
No.of users	cryplib lite	cryptlib cert	cryptlib pro
20	$500	$1,000	$1,500
100	$2,000	$4,000	$6,000
1000	$10,000	$20,000	$30,000
5000+	Please contact cryptlib sales person.		

Server licenses	Please contact cryptlib sales person.
Royalty-based licensing	
Source code license	$20,000
Annual maintenance	15% of runtime license

Sales and Licensing Contacts

International Sales and Licensing

Sean Rudd
Digital Data Security Ltd
St.John Building
1 Beresford St
Auckland



email: sales@datasec.co.nz
Phone +64 9 357-6323 x3301
Fax +64 9 357-6324

Postal address: Digital Data Security Ltd
PO Box 8273
Auckland
New Zealand

US and Canada Sales and Licensing

Stexel Corporation
P.O. Box 431
Smithville
Texas 78957
USA

email: cryptlib@stexel.com
Phone +1 512 237 2757 x210
Fax +1 512 237 2866

Technical Contact

Peter Gutmann

email: pgut001@cs.auckland.ac.nz