

# Cryptonit

<http://cryptonit.org>

Mathias Brossard ([mathias.brossard@idealx.com](mailto:mathias.brossard@idealx.com))

## Abstract

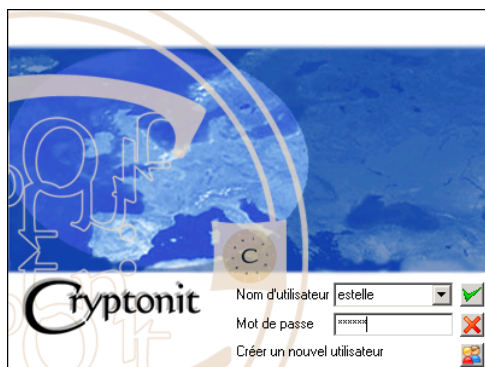


Figure 1: Cryptonit login window

Cryptonit was developed by IDEALX to give everyone access to a simple, secure and free (as in freedom) tool for encryption and signature.

## 1 Introduction

IDEALX has been working on IDX-PKI since 2000. Administrations and corporate accounts contract IDEALX to adapt and integrate IDX-PKI, extending its feature set in the process. Feedback<sup>1</sup> from those customers identified an itch in need of scratching.

Existing client-side cryptographic tools had generally several of the following flaws:

- closed-source,
- proprietary format,
- mono-platform,

<sup>1</sup> Meetings with past and current customers are organised regularly to discuss about project road-maps and exchange experiences.

- based on old code base,
- expensive.

This list of limitations became the starting point for the objectives of Cryptonit.

## 2 Objectives

### 2.1 Open-Source

Security conscious users are suspicious about the origin of security software (and hardware). The nationality of companies or even their shareholders might be considered a serious flaw for security products. Open Source software is less affected, because it is much more easily audited.

### 2.2 Standard compliance

Proprietary editors are fond of using their own undocumented file formats even if similar standards might already exist. The only reason they do that is to "lock-in" their customers. In the past this went as far as using *secret* home-made encryption algorithms.

### 2.3 Multi-platform

Some needs can be solved with thin clients and server based software, but in this case you need client-side software. This means that you will target the Microsoft Windows platform<sup>2</sup>.

<sup>2</sup> Microsoft puts an impressive amount of efforts to keep an ascending compatibility between versions of Windows. Unfortunately, if you intend to support a non-trivial program on more than one version of Windows, you will most probably experience that the compatibility is not perfect.

## 2.4 PKI-oriented

Some solutions are iterations of older password-based encryption software. This generally impacts certificate management (in particular certificate verification) or escrow functions. Cryptonit has been designed from day one with PKI in mind.

## 2.5 Low price

Given the high number of users in administrations and corporate accounts, licensing costs for proprietary software are a big issue. As an example, let's say you have 10.000 users and the security software you want is priced (after negotiation) 100\$ per user. For many organizations this means the project will be put on hold, or limited for a small number of users. In any case, very few manager will pay 10M\$ without looking for alternatives, and we're trying to be one alternative.

## 3 PKI Principles

The software encryption/signature solution Cryptonit is the result of different basic objectives. Cryptonit is an Open Source cross-platform software which enables to use cryptography in all platforms. It's also standards compliant (X.509, PKCS, IETF and RFCs) whereas competitors have developed proprietary file formats (encrypted or signed). These different standards used for Cryptonit development are detailed in the different following parts.

### 3.1 Basic Operations

#### 3.1.1 Symmetric Cryptography

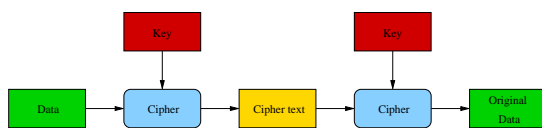


Figure 2: Symmetric Operation

Encryption is the best known process of cryptography: using a key you encrypt the data (from plain-text to cipher-text), and using the same key you can make the inverse operation. In Cryptonit, following symmetric ciphers, all from OpenSSL, are available:

- DES, Triple DES (DES3), DESX
- RC2, RC4, RC5
- Blowfish

- CAST
- IDEA
- AES

Some of these algorithms are be patented in different countries.

#### 3.1.2 Asymmetric Cryptography

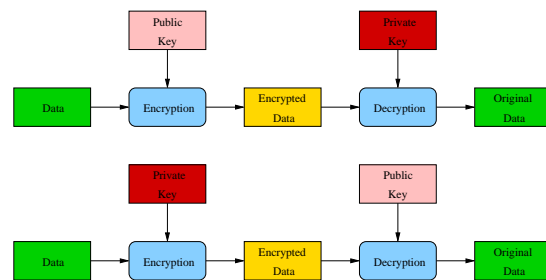


Figure 3: Asymmetric Operation

Whereas symmetric (also known as secret-key) cryptography uses the same key to encrypt and decrypt data, asymmetric (also known as public-key) cryptography uses two keys (a key-pair: both keys are linked): a public key and a private key. The private key is kept secret by the owner of the key. The public key doesn't need to be kept secret (on the contrary). Getting the private key from the public key is supposed to be a *hard* problem, whereas the opposite operation is generally straightforward.

Data is encrypted using the public key and decrypted with the private key. The opposite operation, called digital signature, uses the private key to encrypt and the public key to verify. These properties are important to solve the main problem of symmetric key cryptography: key distribution.

RSA<sup>3</sup> is by far the most commonly used algorithm in asymmetric cryptography. RSA public key operations are much faster than private key operation.

<sup>3</sup>RSA is a reversible public-key cryptosystem for both encryption and signature; it was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman.

### 3.1.3 Applied Asymmetric Encryption

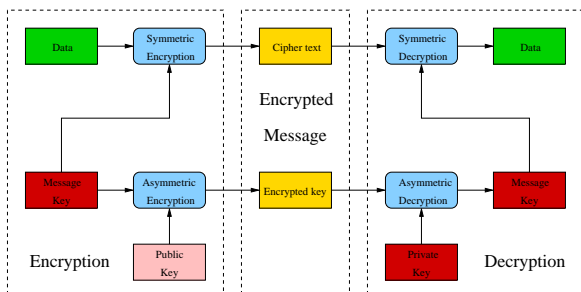


Figure 4: Asymmetric Encryption

With public key encryption you can only encrypt, in one operation, data of the same length as the key. If used for complete documents, this process would be too long. This is why public key encryption is combined with secret key encryption: the data is symmetrically encrypted with a random key (called session key or message key) that is itself encrypted with the public-key. This combination allows to encrypt for multiple recipients without having to encrypt the data several times: only the session key is encrypted for the different recipients.

### 3.1.4 Applied Asymmetric Signature

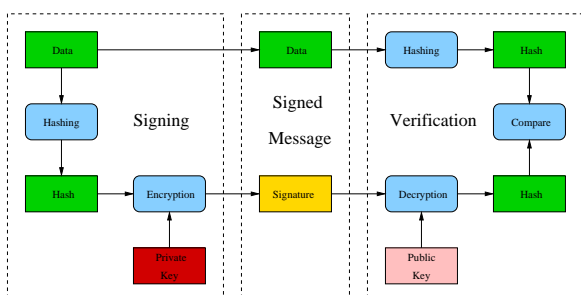


Figure 5: Asymmetric Signature

Similarly as above, the data should not be signed directly: it would be very slow and potentially dangerous (opaque signing). Instead the data is processed using cryptographic hash (or message digest) functions which convert data of arbitrary length into a value of constant length (ex: 160 bits for SHA-1, 128 for MD5) after which only one public key operation is needed.

## 3.2 PKI Principles

### 3.2.1 Certificate

A certificate binds a public/private key pair to an identity. It can be used to authenticate a user (either by signing some protocol dependent data, or by being able to decrypt). Certificates can also be stored in public directories in order for users to send encrypted (using the public key in the certificate) data to the certificate owner.

### 3.2.2 Certification Authority (CA)

Because certificate content could be forged, the role of certification authorities is to securely deliver certificates. These certificates are digitally signed with the private key of the issuing certification authority and can thus be verified with the certificate of the certification authority.

### 3.2.3 Certification hierarchy

The trust in a given PKI is based on a top-level CA (named Root CA). Generally one can imagine the structure as being strict hierarchy of certification authorities (with the Root CA at the top), an inverted tree with the root at the top represents it. The leaves are non-CA PKI entities or simply end-users. This hierarchical trust-model is considered rigid (compared to, for instance, the web-of-trust model from PGP/GPG), but it has the advantages of its simplicity.

### 3.2.4 Certificate Revocation List

If a certificate is not to be trusted anymore (the private key might have been stolen or lost, the owner leaves the organization), it should be revoked<sup>4</sup>. To revoke a certificate, the Certification Authority adds its serial number and date of revocation (other information about the revocation can be added, for instance the reason) to the Certification Revocation List (CRL).

This list is periodically signed and made available to all users. Applications using certificates should verify them using up-to-date Certificate Revocation Lists. A Certificate Revocation List contains validity dates which can be distinct from the actual publication rate: one might choose to publish Certificate Revocation Lists everyday with one week of validity.

Certificate Revocation Lists are generally made available using common protocols HTTP(S), LDAP(S) or FTP. The choice of protocol and location is specified in the Certification Policy of the Certification Authority and sometimes detailed in the certificates themselves using the "CRL Distribution Points" extension.

<sup>4</sup> Not all Certification Authorities publish Certificate Revocation Lists. It is not a requirement in the standards.

### 3.3 PKI Objects and formats

#### 3.3.1 Certificate

The most widely accepted format for certificates is defined by the CCITT X.509 international standard (which Cryptonit supports). This format has the following structure:

- Issuer Distinguished Name: name of the Certification Authority that issued the certificate;
- Serial number: this number identifies the certificate with regards to the issuing certification authority;
- Subject Distinguished Name: name of the certificate owner;
- Validity: starting and ending validity dates of the certificate;
- Public key
- Extensions: a extensible and flexible set of information about the use of the certificate or about the subject of the certificate.
- Certificate signature: seal made by the certification authority to guarantee the integrity.

#### 3.3.2 Private key and Identity

The PKCS#8 format describes private keys. The keys can be protected with a symmetric algorithm whose key is usually derived from a password using the PKCS#5 key derivation algorithm.

The PKCS#12 file format is used for safely store and transmit certificates and private keys (in PKCS#8 format). The PKCS#12 format is protected in confidentiality and integrity.

#### 3.3.3 Certificate request

In order for a user to get a certificate, a certificate request might be generated from the private key, that will be sent to the Certification Authority. The most common format for certificate requests is PKCS#10. This format has the following structure:

- Subject Distinguished Name,
- Public Key,
- Extensions (requested for the certificate, or information for processing the request),
- Signature.

#### 3.3.4 Encrypted / Signed File Format

The PKCS#7 standard defines the Cryptographic Message Syntax (CMS), which allows the exchange of enveloped<sup>5</sup>, encrypted, digested and/or signed objects. PKCS#7 is a widely accepted format, now at the base of several other standards: S/MIME, XML Security, other PKI related standards (TSP, CMC, DVCS, ...), etc.

In Cryptonit we use only enveloped and signed types of containers of PKCS#7. We started using the high-level API of OpenSSL for PKCS#7 support, but we are increasingly changing to lower level functions and structures, in order to add features.

## 4 Technical choices

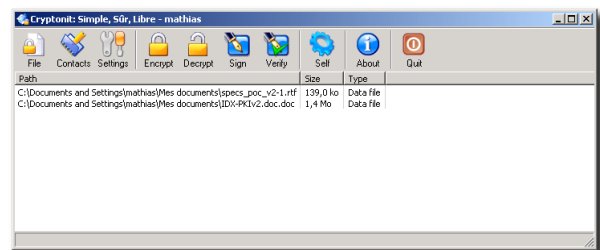


Figure 6: Cryptonit main window

Cryptonit is developed in C++ and under GNU/Linux. The programming language C++ was chosen for an easier object-oriented programming and portability. Cryptonit was built with the help of different open source software packages:

- OpenSSL
- wxWidgets
- OpenLDAP

### 4.1 OpenSSL

OpenSSL is a library which implements the Secure Socket Layer (SSL) also known as Transport Layer Security (TLS). OpenSSL is used by the Cryptonit cryptographic core and performs almost all cryptographic operations in Cryptonit and most file format handling. OpenSSL includes ASN.1<sup>6</sup> routines used for manipulating the PKI and PKCS formats: X.509 Certificates

<sup>5</sup>Enveloped data is the combination of symmetrically encrypted data with the key asymmetrically encrypted for one or more recipients.

<sup>6</sup>Abstract Syntax Notation One (ASN.1) is a formal language for abstractly describing messages to be exchanged between distributed computer systems. ASN.1 is used as a basis for many international standards.

Graphic interface: wxWidgets	
Namespace Cryptonit	
OpenSSL	OpenLDAP

Table 1: Cryptonit architecture

and Certificate Revocation Lists, PKCS#7 Cryptographic Messages, PKCS#8 Private Keys, PKCS#10 Certificate Requests, etc.

## 4.2 wxWidgets

The graphic interface brings conviviality in software use. Cryptographic operations appear easier which is essential for data security. wxWidgets is a quite mature cross-platform library C++ for GUI (Graphic User Interface) development. Cryptonit is created for different operating system with the same source code. It uses the platform look and feel where the software is performing (Gtk2 under Linux/Unix, Cocoa under Mac OS X, Win32 under Windows). wxWidgets is published under the open source wxWidgets Library license (a relaxed version of L-GPL).

## 4.3 OpenLDAP

The library OpenLDAP gives Cryptonit access to LDAP directories in order to retrieve user entries with certificates into the addressbook, or Certificate Revocation List from Certification Authority entries.

## 4.4 Architecture

Cryptonit is divided in two main parts: graphic interface and cryptographic core which contains encryption/decryption, signature, LDAP request, multi-user part with authentication and certificate management methods. This architecture improves Cryptonit integration to an OS. It could be also integrated to applications which use PKI certificates (e-mails, sharing files and word processing client, or other data entry forms management system).

Cryptonit enables people to easily perform cryptographic operations. These operations are based on PKCS#7 format implemented by the library OpenSSL.

## 4.5 A cross-platform software

Cryptonit is easily integrated in business infrastructure. This cross-platform software can be deployed in large scale. The supported platforms are:

- Microsoft Windows 9x/NT/2000/XP

- GNU/Linux

- Mac OS X

## 5 Certificate management

### 5.1 Certificate request

The certificate request is made from a graphic interface in which the user fills information as his name, telephone number, address, key size, the symmetric encryption algorithm which protects his public/private key pair. After that operation, a public/private key is generated and a certificate X509 request is filled with user data. Whole data are exported into a PKCS#10 file. Then the user sends the file to the RA (Registration Authority) which will valid or reject the request. Cryptonit saves a request copy in a directory with the private key in order to integrate them when certificate will be signed and received. So the program verifies the correspondence between the certificate and the request then between the private key and the public key. If it is all right, a PKCS#12 file is generated with the signed certificate and the private key.

### 5.2 Certificate verification

One of the important features of Cryptonit is certificate verification. When verifying a signature or before encrypting a document, the certificate has to be verified. In order that a certificate might be validated, the certificate chain has to be valid which means all parent certificates are recursively validated.

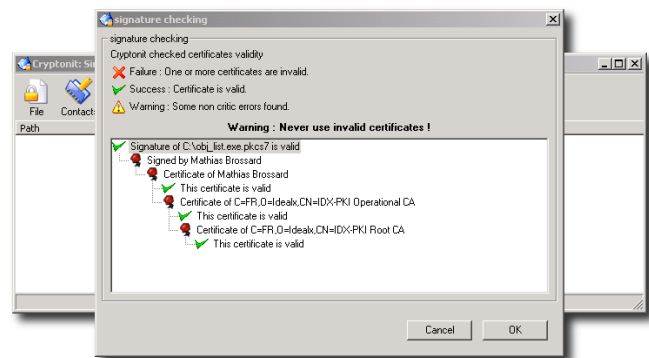


Figure 7: Signature and certificate verification

At each level of the chain the following checks are enforced.

- Certificate signature,

- Certificate key usage extension with respect to Certification Authority function,
- Time validity,
- Certificate Revocation List freshness,
- Certificate Revocation List signature,
- Certificate revocation status.

### 5.3 Identity management

In Cryptonit, you can have several identities within the same account. Identities can be used for different usage (one for signature and one for encryption) or in relation to different contexts (for instance, a user is allowed to use one identity for external messages and one for internal messages). An identity translates into a certificate and its private key. When using the software key-store, each identity is stored inside a PKCS#12 file.

### 5.4 Certification authorities and CRL management

When an accounting is created or an identity is added in Cryptonit certification authorities certificates are imported which are sometimes in the PKCS#12 file. But it's possible to add ex post facto CA certificates for which a CRL can be downloaded.

## 6 Useful features

### 6.1 Multi-user

Cryptonit was developed to support multi-user management independently of OS support. The same user could possess several accounts in the software, in the case of a computer shared among several persons. Still, Cryptonit allows to use several identities under the same account. User data are saved in the home directory of the system user launching the application ("/home" under Unix and "C:\Documents and settings" under Windows 2000) including address-book, certification authorities, identities, certificate revocation lists.

### 6.2 Address-book

Cryptonit offers an address-book to save contacts and their certificates which are needed to send encrypted files. Entries can be added manually (importing certificate from local files) or from an LDAP.

### 6.3 Single binary

In order to simplify the use of Cryptonit under Windows, we package Cryptonit as a single binary. This means there are no need for administrator privileges to use Cryptonit, you only need to copy the executable file.

### 6.4 Internationalization

Cryptonit has been developed to be used by everyone. That's why it supports different languages (French, English, German, Brazilian Portuguese, Dutch). We use the Open Source gettext in order to simplify i18n handling. Binary translation files are included in the program, and unpack automatically into the users Cryptonit directory. We needed to do that in order to support the 'single binary' distribution.

### 6.5 Smart-cards

Smart-cards enable user to keep secrets in a cryptographic support. These physically and logically personalized cards protect private keys and other secrets controlling their access with PIN codes. Smart-cards are accessible through a standardized API called PKCS#11. This flexible 'C' API allows the development of portable, vendor-neutral security applications using cryptographic smart-cards (or other cryptographic devices). Cryptonit supports PKCS#11 through a key-store class abstraction.

### 6.6 Self-decrypting files

Sometimes it might be necessary to send encrypted files to people who don't have a certificate and possibly not the ability to install software. For the cases when certificate encrypted files is not an option, we have included the self-decrypting file generation. The data is encrypted with AES-256, and the session key is derived from the password using the PKCS#5 algorithm with SHA-256 as the hash-function.

At this moment, only Windows and Linux (x86) platforms are available. The system works by concatenating a precompiled program (stub) and the encrypted data. The program when launched will find the encrypted data, and given the right password will decrypt the data.

In terms of security, it might not be a totally safe and scalable solution, because of the transmission of the password and the fact that you execute a program coming from a probably unsecured channel. Still it's a useful feature.

## 7 Work in progress

The roadmap for Cryptonit includes (in no particular order):

- Encrypted volumes: Cryptonit is at the moment able to encrypt files, the next step is to encrypt entire volumes. Work is being done for Linux and Windows platform: this involves, in both cases, writing some kernel-space code.
- Time-stamp: Time-stamping is an essential element in non-repudiation services, because it allows to prove that a certain data existed before a particular time. It works with a client requesting time-stamp for a data (generally the hash of the data, rather than the data itself) and the server returns the data (or hash) signed together with the time of the operation (coming from a trusted time reference). The RFC3161 defines the protocol for implementing time-stamp within PKIX.
- OCSP (Online Certificate Status Protocol) Validation: Certificate validity is a complex operation because the certification chain (CA certificates, revoked certificates) must be verified. Then verification is delegated to a verification server defined by the IETF RFC2560. OCSP is a request/response protocol to obtain on-line revocation information from a trusted entity (OCSP responder).
- File wiping: In most operating systems when a file is deleted, its content is not erased until overwritten (there are some methods to recover overwritten data). The file name doesn't appear in the disk index but the data still remains on the disk.
- XML Signature and XML Encryption support: XML files can be signed and encrypted in Cryptonit using the PKCS#7 format as it can handle any binary data. Supporting the XML formats would allow to integrate security into XML applications or communications.
- etc.

## References

- [RFC2459] IETF. Internet X.509 Public Key Infrastructure Certificate and CRL Profile — RFC 2459 (<http://www.ietf.org/rfc/rfc2459.txt>).
- [RFC2560] IETF. X.509 Public Key Infrastructure Online Certificate Status Protocol (OCSP) — RFC 2560 (<http://www.ietf.org/rfc/rfc2560.txt>).
- [RFC2633] IETF. S/MIME Version 3 Message Specification — RFC 2633 (<http://www.ietf.org/rfc/rfc2633.txt>).
- [RFC3161] IETF. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP) — RFC 3161 (<http://www.ietf.org/rfc/rfc3161.txt>).
- [PKCS#1] RSA Labs. PKCS #1 — RSA Encryption Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2125>).
- [PKCS#5] RSA Labs. PKCS #5 — Password-Based Encryption Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2127>).
- [PKCS#7] RSA Labs. PKCS #7 — Cryptographic Message Syntax Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2129>).
- [PKCS#8] RSA Labs. PKCS #8 — Private-Key Information Syntax Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2130>).
- [PKCS#11] RSA Labs. PKCS #11 — Cryptographic Token Interface Token Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2133>).
- [PKCS#12] RSA Labs. PKCS #12 — Personal Information Exchange Syntax Standard (<http://www.rsasecurity.com/rsalabs/node.asp?id=2138>).