Crypto with OpenSSL

GUAN Zhi
guanzhi@infosec.pku.edu.cn

Outline

- *OpenSSL Programing*
- *Program secure code tips*
- *CPK version 0.6.8 source code organization*
- *CPK version 0.7 new features*

OpenSSL Programming

What is OpenSSL?

- Cryptography tool kit.
- Open source implementation of SSL v2/v3 and TLS v1.
- PKI/CA, cryptographic command line tools.

OpenSSL Includes

- *Source code, http://www.openssl.org/source/ (release) or ftp://ftp.openssl.org/snapshot/ (snapshot)*
- *Include header files, #include <openssl/des.h> or /usr/src/include/openssl/*
- *libraries, libeay32.[lib|dll], ssleay32.[lib/dll] (Win32) or libcrypto.[so|a], libssl.[so|a] (Linux).*
- *executable binary: openssl[.exe]*

Command Line Tool

- The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for
  - – Creation of RSA, DH and DSA key parameters
  - – Creation of X.509 certificates, CSRs and CRLs
  - – Calculation of Message Digests
  - – Encryption and Decryption with Ciphers
  - – SSL/TLS Client and Server Tests
  - – Handling of S/MIME signed or encrypted mail

Supported PKI Standards

- ASN.1 encoding
- SSLv2, SSLv3, TLSv1
- PKCS #5, PKCS #7, PKCS #8, PKCS #12, X.509v3
- OCSP
- PEM

Supported Algorithms

- *Block ciphers: AES, DES, 3DES, Blowfish, Camellia, CAST, Idea, RC2, RC5.*
- *Block cipher modes: ECB, CBC, CFB, OFB, CTR ...*
- *Stream ciphers: RC4*
- *Digest algorithms: MD2, MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, Ripemd-160.*
- *MAC: HMAC, MDC2*
- *Public key crypto-systems: DH, DSA, RSA, ECC.*

OpenSSL License

- OpenSSL is licensed under Apache style license, free to get and use it for commercial and non-commercial purposes subject to some simple license conditions.
- Redistributions of any form what so ever must retain the following acknowledgment:
  - "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

Download, Build and Install

- $ ./config
- $ make
- $ make test
- $ make install

- Download the source code from *official* openssl homepage.
- The *make test* step is very important! In some platforms this step may fail.

Source Code

- *openssl/apps/ openssl command line tool*
- *openssl/crypto/ libcrypto crypto library*
- *openssl/ssl/ libssl SSL/TLS library*
- *openssl/demos/ some examples*
- *openssl/docs/ man pages and howtos*
- *openssl/engines/ hardware crypto accelerator drivers*
- *openssl/include/ include header files*

Source Code

- *openssl/MACOS,ms,Netware,os2,VMS/ platform specific*
- *openssl/test/ code for make test, important.*
- *openssl/times/ code for ``openssl speed'' benchmark*
- *openssl/tools/*
- *openssl/util/ perl shells for C code generation*

openssl/crypto/*

Ciphers
- aes
- bf
- camellia
- cast
- des
- idea
- rc2
- rc4
- rc5

Hash Algors and MACs
- hmac
- md2
- md4
- md5
- mdc2
- ripemd
- sha

Public Key Crypto
- bn
- dh
- dsa
- dso
- ec
- ecdh
- ecdsa
- rsa

PKI
- asn1
- krb5
- objects
- ocsp
- pem
- pkcs7
- pkcs12
- x509
- x509v3

Data Structure
- buffer
- lhash
- pqueue
- stack
- store
- txt_db

Utilities
- bio
- engine
- err
- evp
- threads

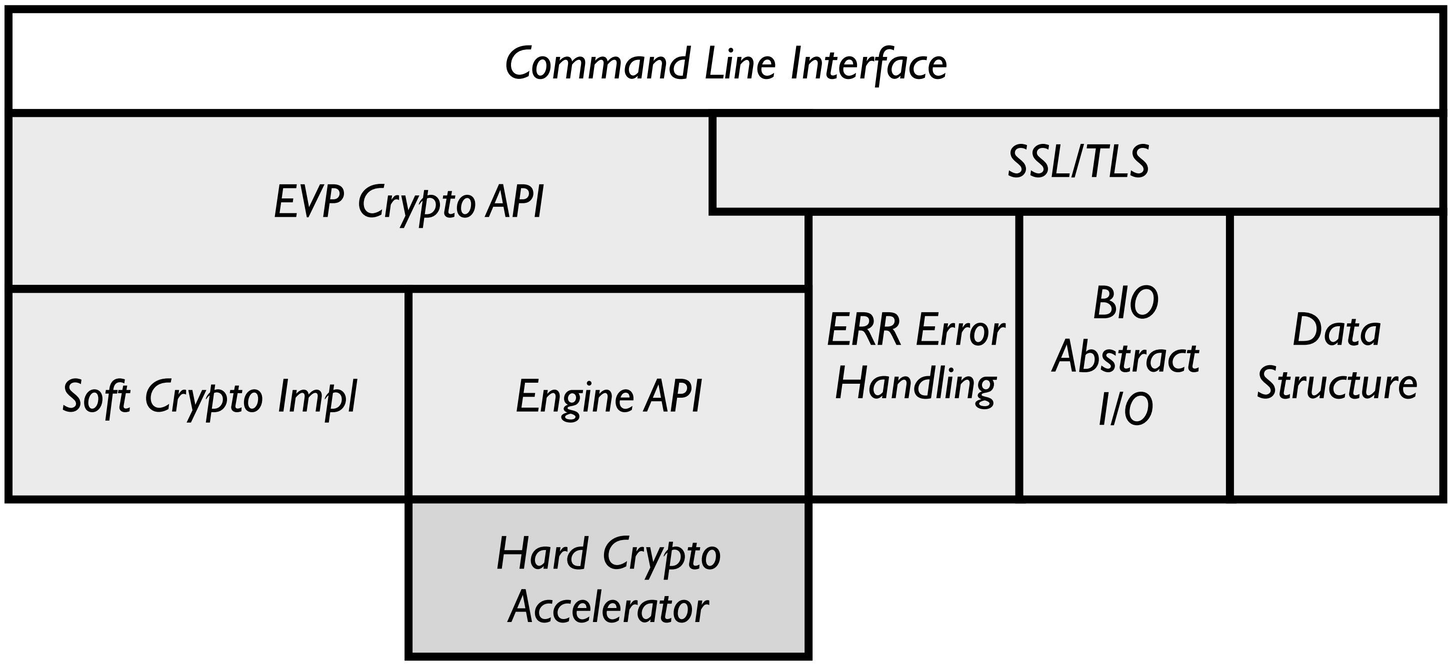
RNG
- seed
- rand

CLI
- ui
- conf

SSL/TLS
- ssl

Compression
- comp

Architecture of OpenSSL



Crypto API

- Microsoft Crypto API
- RSA PKCS #11: Cryptographic Token Interface Standard
- OpenGroup Common Security: CDSA and CSSM
- OpenSSL EVP interface
  - Change different cipher algorithms without change the source code.

EVP Symmetric Encryption

```
#include <openssl/evp.h>

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
const EVP_CIPHER *cipher = EVP_aes_128_cbc();

EVP_EncryptInit(&ctx, cipher, key, iv);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
...
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptFinal(&ctx, out, &outlen);

EVP_CIPHER_CTX_cleanup(&ctx);
```

EVP Symmetric Encryption

```
#include <openssl/evp.h>

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
const EVP_CIPHER *cipher = EVP_aes_128_cbc();

EVP_EncryptInit(&ctx, cipher, key, iv);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
...
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptFinal(&ctx, out, &outlen);

EVP_CIPHER_CTX_cleanup(&ctx);
```

EVP Symmetric Encryption

```
#include <openssl/evp.h>

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
const EVP_CIPHER *cipher = EVP_aes_256_ofb();

EVP_EncryptInit(&ctx, cipher, key, iv);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
...
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptFinal(&ctx, out, &outlen);

EVP_CIPHER_CTX_cleanup(&ctx);
```

EVP Symmetric Encryption

```
#include <openssl/evp.h>

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
const EVP_CIPHER *cipher = EVP_aes_128_cbc();

EVP_EncryptInit(&ctx, cipher, key, iv);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
...
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptFinal(&ctx, out, &outlen);

EVP_CIPHER_CTX_cleanup(&ctx);
```

EVP Symmetric Decryption

```
#include <openssl/evp.h>

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
const EVP_CIPHER *cipher = EVP_aes_128_cbc();

EVP_DecryptInit(&ctx, cipher, key, iv);
EVP_DecryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_DecryptUpdate(&ctx, out, &outlen, in, inlen);
...
EVP_DecryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_DecryptFinal(&ctx, out, &outlen);

EVP_CIPHER_CTX_cleanup(&ctx);
```

EVP With Engine

```
#include <openssl/evp.h>

EVP_CIPHER_CTX ctx;
EVP_CIPHER_CTX_init(&ctx);
const EVP_CIPHER *cipher = EVP_aes_128_cbc();

EVP_EncryptInit_ex(&ctx, cipher, engine, key, iv);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
...
EVP_EncryptUpdate(&ctx, out, &outlen, in, inlen);
EVP_EncryptFinal_ex(&ctx, out, &outlen);

EVP_CIPHER_CTX_cleanup(&ctx);
```

EVP_CIPHER Interface

```
struct evp_cipher_st {
    int nid;
    int block_size;
    int key_len;
    int iv_len;
    unsigned long flags;
    int (*init)(EVP_CIPHER_CTX *ctx, const unsigned char *key, const unsigned char *iv, int enc);
    int (*do_cipher)(EVP_CIPHER_CTX *ctx, unsigned char *out, const unsigned char *in, unsigned int inl);
    int (*cleanup)(EVP_CIPHER_CTX *);
    int ctx_size;
    int (*set_asn1_parameters)(EVP_CIPHER_CTX *, ASN1_TYPE *);
    int (*get_asn1_parameters)(EVP_CIPHER_CTX *, ASN1_TYPE *);
    int (*ctrl)(EVP_CIPHER_CTX *, int type, int arg, void *ptr);
    void *app_data;
} EVP_CIPHER;
```

EVP_CIPHER Interface

```
struct evp_cipher_st {
	int nid;
	int block_size;
	int key_len;
	int iv_len;
	unsigned long flags;
	int (*init)(EVP_CIPHER_CTX *ctx, const unsigned char *key, const unsigned char *iv, int enc);
	int (*do_cipher)(EVP_CIPHER_CTX *ctx, unsigned char *out, const unsigned char *in, unsigned int inl);
	int (*cleanup)(EVP_CIPHER_CTX *);
	int ctx_size;
	int (*set_asn1_parameters)(EVP_CIPHER_CTX *, ASN1_TYPE *);
	int (*get_asn1_parameters)(EVP_CIPHER_CTX *, ASN1_TYPE *);
	int (*ctrl)(EVP_CIPHER_CTX *, int type, int arg, void *ptr);
	void *app_data;
} EVP_CIPHER;
```

EVP Digest

```
#include <openssl/evp.h>

EVP_MD_CTX ctx;
EVP_MD_CTX_init(&ctx);
const EVP_MD *md = EVP_sha1();

EVP_DigestInit(&ctx, md);
EVP_DigestUpdate(&ctx, in, inlen);
EVP_DigestUpdate(&ctx, in, inlen);
...
EVP_DigestUpdate(&ctx, in, inlen);
EVP_DigestFinal(&ctx, digest, &digest_len);

EVP_MD_CTX_cleanup(&ctx);
```

EVP_MD

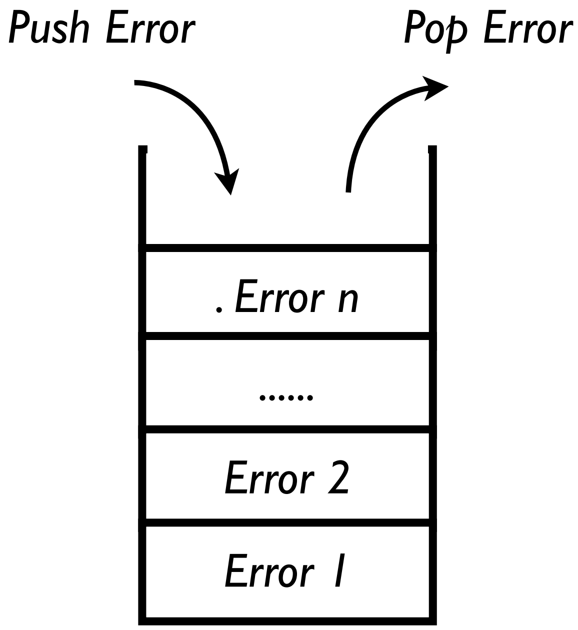
```
struct env_md_st {
        int type;
        int pkey_type;
        int md_size;
        unsigned long flags;
        int (*init)(EVP_MD_CTX *ctx);
        int (*update)(EVP_MD_CTX *ctx,const void *data,size_t count);
        int (*final)(EVP_MD_CTX *ctx,unsigned char *md);
        int (*copy)(EVP_MD_CTX *to,const EVP_MD_CTX *from);
        int (*cleanup)(EVP_MD_CTX *ctx);
        int required_pkey_type[5];
        int block_size;
        int ctx_size;
} EVP_MD;
```

EVP Public Key API

EVP_SignInit_ex()
EVP_SignInit()
EVP_SignUpdate()
EVP_SingFinal()

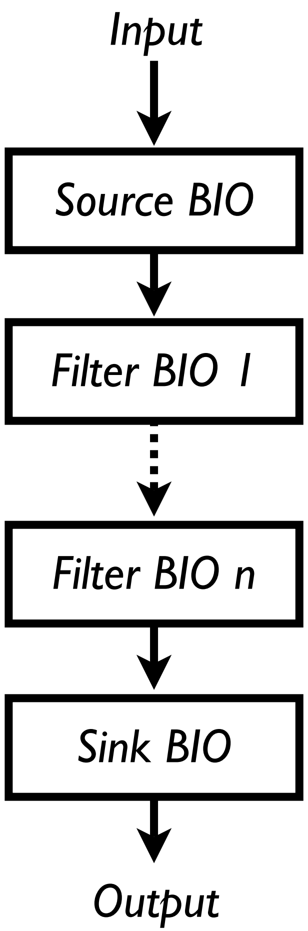
EVP_VerifyInit_ex()
EVP_VerifyInit()
EVP_VerifyUpdate()
EVP_VerifyFinal()

Error Stack



Error Stack

BIO: Abstract IO Interface



BIO_s_mem(),

BIO_s_file(), BIO_s_fd()

BIO_s_socket(), BIO_s_accept(), BIO_s_connect()

BIO_s_bio(), BIO_s_null()

BIO_f_base64(), BIO_f_buffer(), BIO_f_cipher()

BIO_f_md(), BIO_f_null(), BIO_f_ssl()

BIO Example, Base64

```
BIO *bio, *b64;
char message[] = "Hello World \n";

b64 = BIO_new(BIO_f_base64());
bio = BIO_new_fp(stdout, BIO_NOCLOSE);
bio = BIO_push(b64, bio);
BIO_write(bio, message, strlen(message));
BIO_flush(bio);

BIO_free_all(bio);
```

Program Secure Code

Random Numbers

- *Random numbers are widely used in cryptography:*
  - *public key generation, symmetric encryption keys, MAC keys, symmetric encryption initial vector (IV)salt, nonce*
- *Random numbers must be generated from Cryptographic Random Number Generator (RNG), not C rand() function!*
- *OpenSSL RNG,*

Random Numbers (cont.)

- TRNG service from operating system, CryptGenRandom
  - /dev/random on Linux (NOT /dev/urandom)
  - CryptGenRandom() on Windows
- TRNG service from hardware device, such as from USB tokens, crypto accelerators and smart cards through PKCS #11 or MS Crypto API interface.
- TRNG service from OpenSSL rand interface.

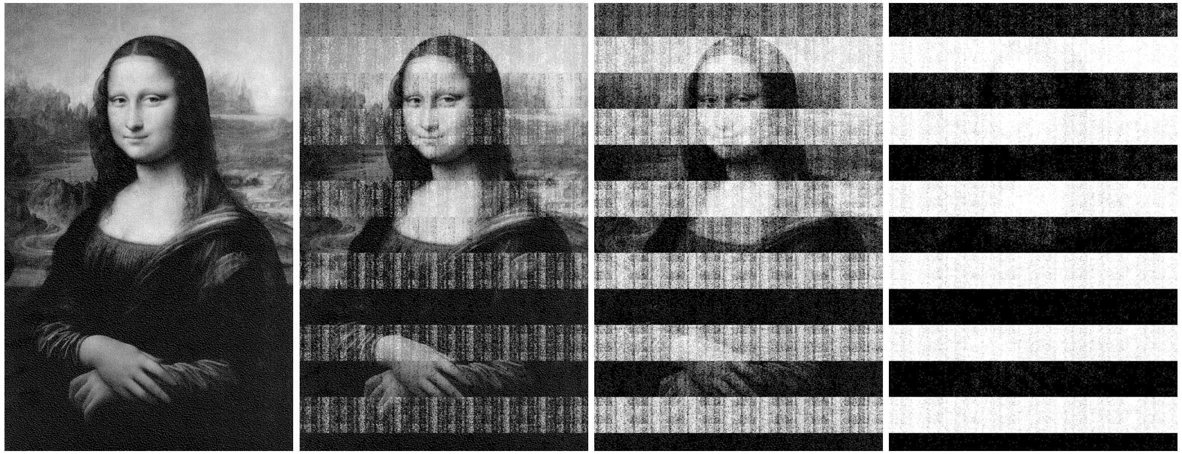
Recommended Key Length

- *Symmetric encryption, AES, 128-bit*
- *Digest algorithm, SHA-256*
- *HMAC key, same to digest algorithm, 256-bit*
- *RSA 1024-bit or 2048-bit*
- *ECC 192-bit*

Keep Secrets in Memory

- *Never hard code secrets in a program.*
- *Secret data of one process/user may be read by other process or user. The live time of a session key should be as short as possible.*
- *Data may be swapped to disk. Use system calls to lock the key's memory.*
- *Data remain in memory even after the RAM is powered off for 5 minutes. After the cryptographic operation, the keys should be securely cleaned from the memory.*

Frozen Attack



after 5 seconds 30 seconds 60 seconds 5 minutes

Lock and Clean Memory

- Preventing memory from being paged to disk, mlock() on Linux, and VirtualLock() on Windows.
- Erasing data from memory securely and as soon as possible, use handwrite clean_memory() instead of standard libc memset().

```
volatile void clean_memory( volatile void *dst, size_t len )
{
    volatile unsigned char *p;
    for (p = (volatile unsigned char *)dst; len; p[--len] = 0)
        ;
}
```

Keep Secrets on Disk

- *Never write the plaintext secrets to disk.*
- *Use cryptographic tools to encrypt the secrets with a password or a public key/certificate.*

Protect Secret Files

- Protect secret files with symmetric key cryptography.
  - Standard: PKCS #5 password based encryption and MAC.
  - Tools: GnuPG, OpenSSL command line tool.
- Protect secret files with public key cryptography.
  - Standard: PKCS #7, PKCS #12
  - Tools: GnuPG

With Tools

- *GunPG, an open source implementation of PGP.*
    - *gpg -c plaintext.txt*
- *TrueCrypt, Disk encryption*
- *OpenSSL (not recommended).*

Wipe a File

- It's hard to destroy a file on disk, even impossible.
- For some file systems, files are never deleted.
- A deleted file can be recovered even after the sector is written 23 times.
- Some tools:
  - gnupg, Linux
  - PGP, Windows
  - ……

Encrypt with Password

- Select a secure password.
- Check the validity of this password.
- Derive a encryption key and a MAC key from the password.
- Encrypt the plaintext with the encryption key.
- Generate a HMAC of the plaintext with the MAC key.
- Clean the plaintext, keys and password.

Deriving Keys from a Password

```
#include <evp.h>
#include <openssl/rand.h>

char *passwd = "secret password";
unsigned char salt[8];
unsigned char iv[16];
int iter = 65535;
unsigned char key[16];
RAND_bytes(salt, sizeof(salt));
RAND_bytes(iv, sizeof(iv));
PKCS5_PBKDF2_HMAC_SHA1(passwd, strlen(passwd),
    salt, sizeof(salt), iter, sizeof(key), key);

AES_KEY aes_key;
AES_set_encrypt_key(key, 128, aes_key);
// AES encrypt routines ...
```

Serialization Code Generation

```
typedef struct privkey_st {
        long                     ver;
        ASN1_OBJECT         *ecoid;
        ASN1_UTF8STRING *matrixid;
        ASN1_UTF8STRING *keyid;
        ASN1_INTEGER        *keydata;
} PRIVKEY;

ASN1_SEQUENCE(PRIVKEY) = {
        ASN1_SIMPLE(PRIVKEY, ver, LONG),
        ASN1_SIMPLE(PRIVKEY, ecoid, ASN1_OBJECT),
        ASN1_SIMPLE(PRIVKEY, matrixid, ASN1_UTF8STRING),
        ASN1_SIMPLE(PRIVKEY, keyid, ASN1_UTF8STRING),
        ASN1_SIMPLE(PRIVKEY, keydata, ASN1_INTEGER),
} ASN1_SEQUENCE_END(PRIVKEY);
IMPLEMENT_ASN1_FUNCTIONS(PRIVKEY);
```

CPK 0.6.8

- A command line tool, written in C, base on OpenSSL.
- Provide CPK system setup, identity-based public key encryption/decryption, digital signature generation and verification.
- Support data types serialization, encoding with ASN.1 syntax and DER format.
- The source code style is similar to openssl.

Elliptic Curve Cryptography

- ECC basic types:
  - Private key, big integer, BIGNUM
  - Public key, elliptic curve point, EC_POINT
  - Elliptic curve parameters, EC_GROUP
- ECC algorithms:
  - ECDH, Elliptic Curve Diffie-Hellman Key Exchange
  - ECDSA, Elliptic Curve Digital Signature Algorithm
  - ECIES, Elliptic Curve Integrated Encryption Scheme

Select an Elliptic Curve

- *OpenSSL embeds a set of elliptic curves.*
  - *EC_GROUP *ec = EC_GROUP_new_by_curve_name(OBJ_txt2nid("secp192k1"));*

Map Identity to Matrix Indexes

- Use a Key Derive Function (KDF) to derive a fixed length output from a variable length input.
- typedef void *(*KDF)(const void *in, size_t inlen, void *out, size_t *outlen);
- X9.63 KDF
  - x9_63_kdf_sha1()
  - x9_63_kdf_sha256()
  - ....

Setup

- *The authority generates a pair of matrix, at first the private matrix, and the corresponding public matrix from the private matrix. The public matrix is published publicly, and the private matrix is kept secretly inside the authority.*
  - *PRIVMATRIX_create() ⇒ PRIVMATRIX*
  - *PRIVMATRIX_gen_pubmat(PRIVMATRIX) ⇒ PUBMATRIX*
- *Given a user's identity string, the authority generates the user's private key.*
  - *PRIVMATRIX_gen_privkey(PRIVMATRIX, ID) ⇒ PRIVKEY*

Encrypt and Decrypt

- *The sender takes recipient's identity string and the public matrix to encrypt a message.*
  - *PUBMATRIX_encrypt(PUBMATRIX, ID, PlainText) ⇒ RCPTINFO*
- *The recipient use his private key to decrypt the encrypted message.*
  - *PRIVKEY_decrypt(PRIVKEY, RCPTINFO) ⇒ PlainText*

Sign and Verify

- The sender takes his private key to generate a signature of a message.
  - PRIVKEY_sign(PRIVKEY, MessageDigest) ⇒ SIGINFO
- The recipient use sender's identity string and the public matrix to verify the signature of the message.
  - PUBMATRIX_verify(PUBMATRIX, MessageDigest, SIGINFO) ⇒ Yes/No

Serialization

- Serialization data types to/from memory
  - *int i2d_TYPE(TYPE *, unsigned char **);*
  - *int d2i_TYPE(TYPE *, const unsigned char **, int);*
- Serialization data types to/from files
  - *int i2f_TYPE(TYPE, filename);*
  - *int f2i_TYPE(TYPE, filename);*

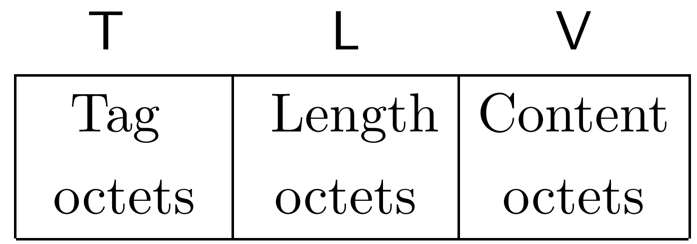
Data Structures

- *PUBMATRIX the public matrix*
- *PRIVMATRIX the private matrix*
- *ASN1_UTF8STRING the public key, aka, identity string, this is a data type provided by OpenSSL.*
- *PRIVKEY the private key, with key owner's information*
- *SIGINFO generated signature, with signer's information*
- *RCPTINFO encrypted data, with recipient's information. This encrypted data should only be short data, e.g. keys.*

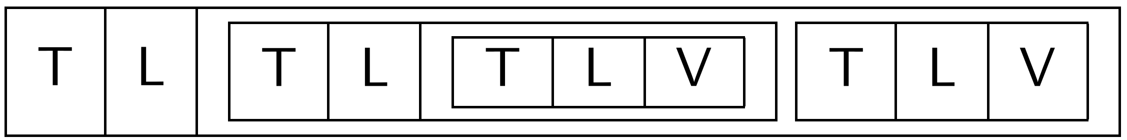
Related Techniques

- ASN.1 encoding
- TLV: Type-Length-Value
- DER encoding
- openssl asn1parse

TLV



(a) Triplet TLV



(b) Recursive principle

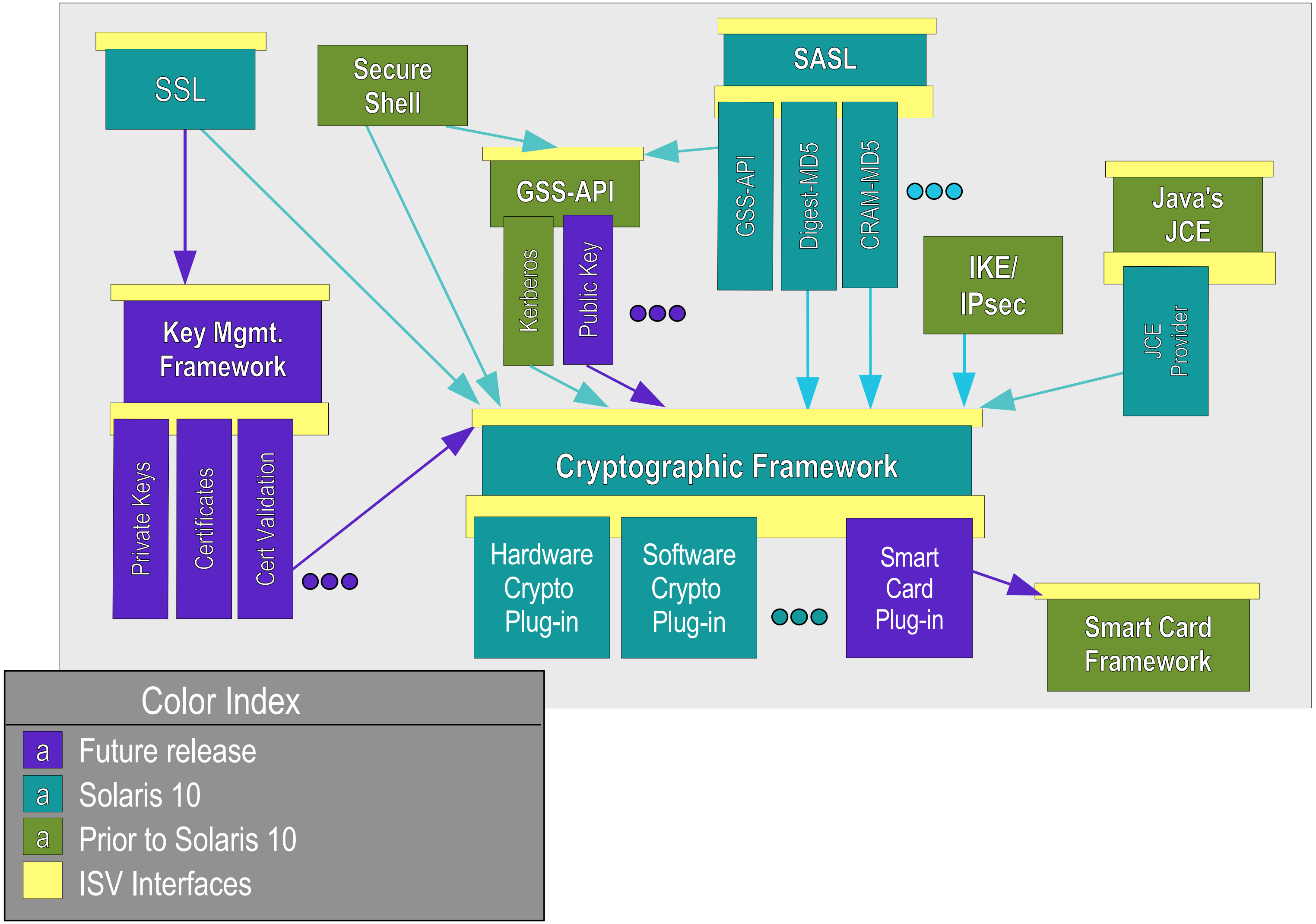
CPK Crypto Library 0.7

- PKCS #11 API supported with thread safe.
- PKCS #7 crypto message syntax standard supported.
- SECG ver.1.7 ECC bulk encryption supported.
- ASN.1 and DER encoding supported.
- OpenSSL buffered IO, include memory, file, socket ...
- OpenSSL error stack supported.
- All platforms supported, Linux, Solaris, Windows, Mac.

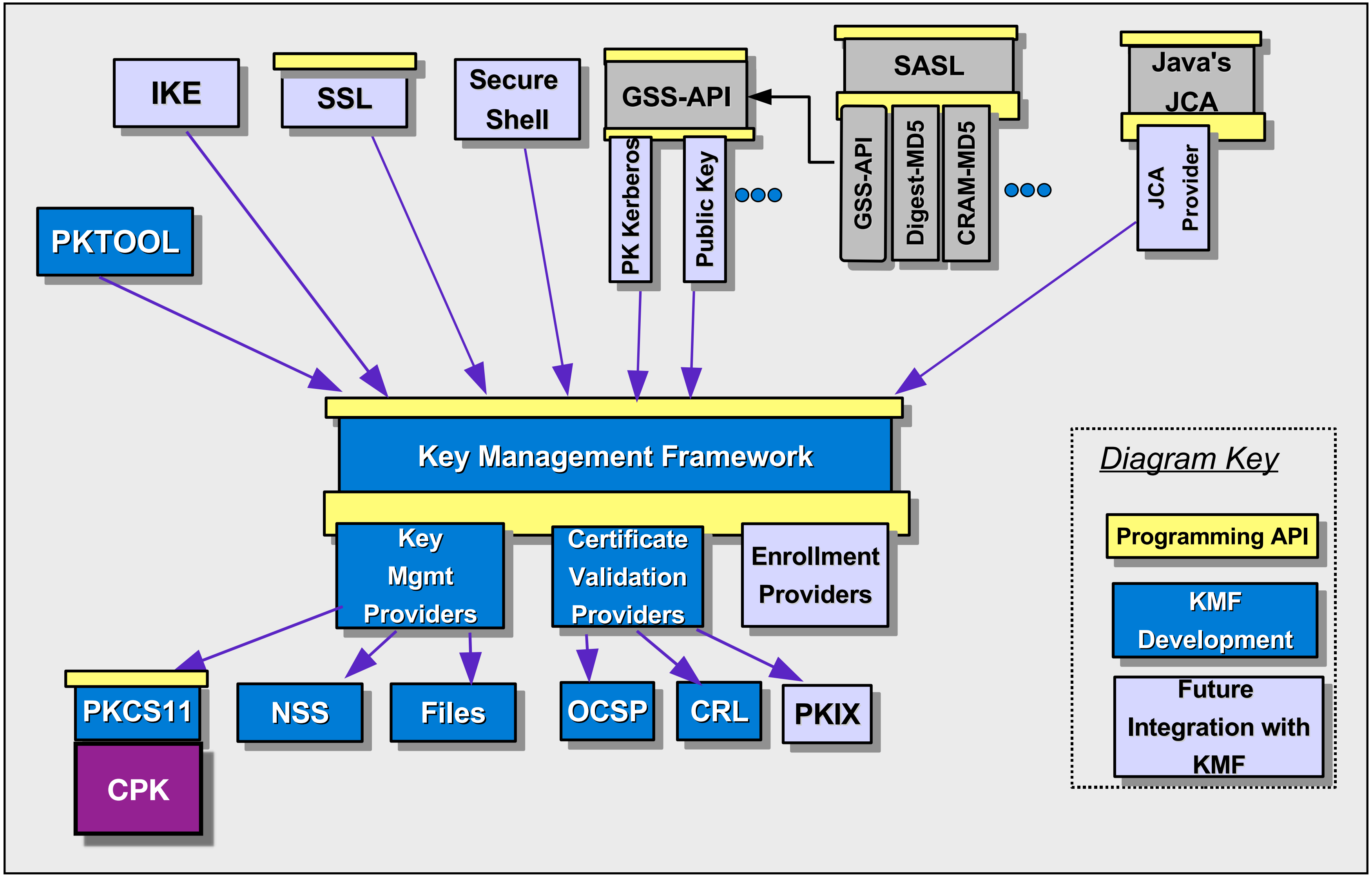
CPK with OpenSolaris

- Solaris, the most secure operation system with great security features, include cryptography framework, key management framework, filesystem encryption and hardware crypto accelerator.
- Solaris Crypto Framework (SCF) supports: extensible cryptographic interfaces, vendor hardware and software, default supports AES, 3DES, RSA, ECC, SHA-1, and CPK.
- Free and open source.

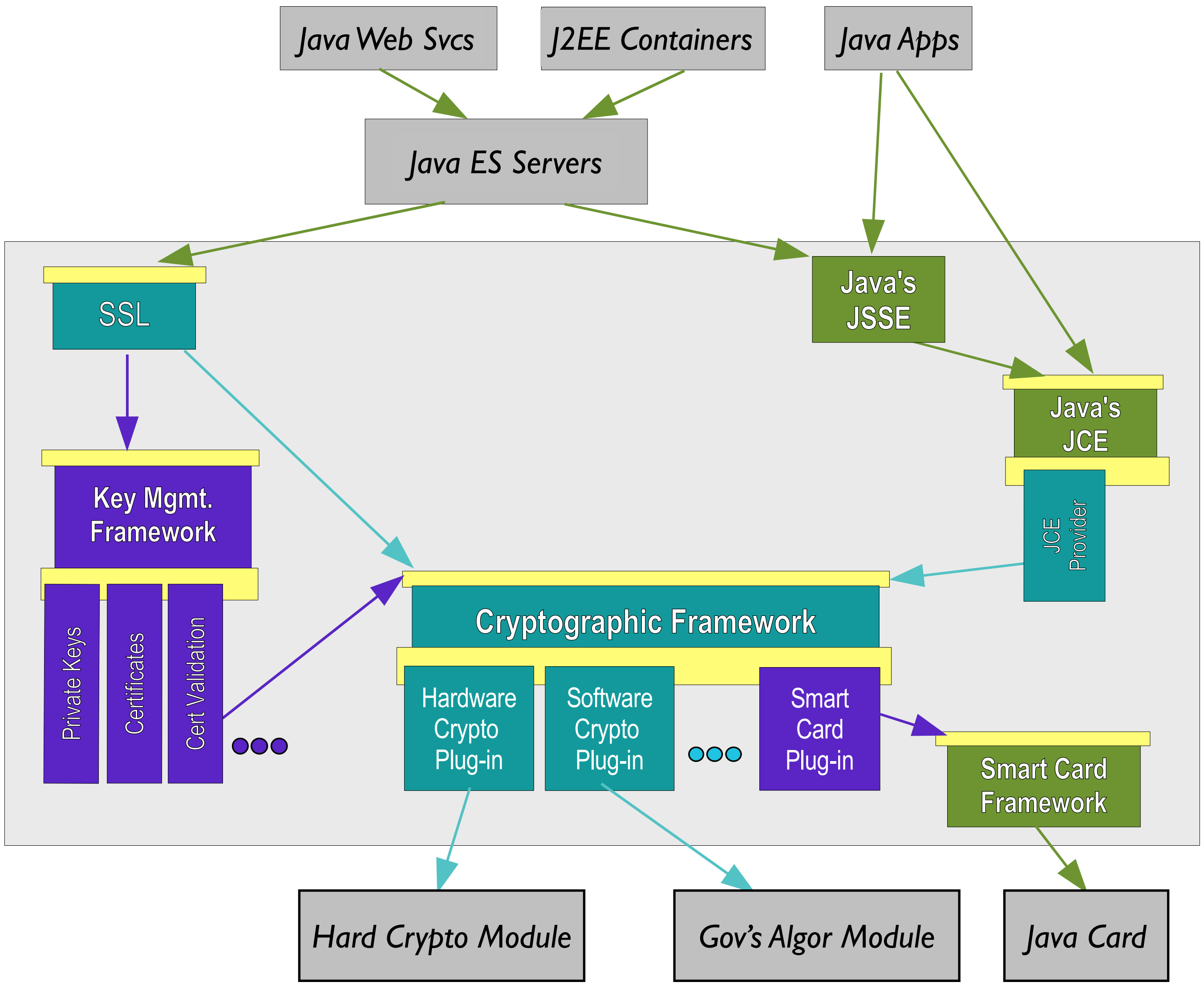
Solaris Crypto Framework



Key Management Framework



Web Services Security Platform



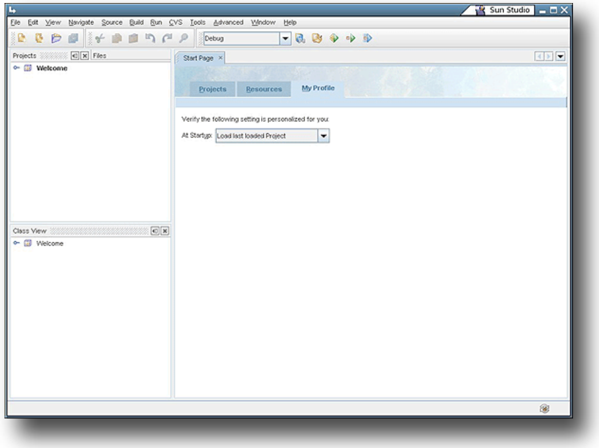
ZFS Filesystem Encryption

- *Support keys and crypto operations in hardware.*
- *Support local (USBKey, smart card, TPM, password) or remote key manager.*
- *Support secure delete by "key destruction"*
- *Confidentiality : All application data, POSIX layer data (permissions, owner) and directory structure are encrypted with AES in CCM/GCM mode.*
- *Integrity: integrity protection of data and metadata by Fletcher or SHA256.*

Crypto Accelerator Board

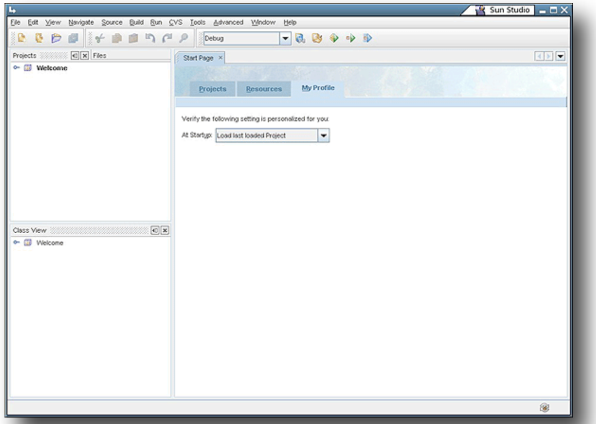
- About 13,000 operations per second with 1,024 bit modular exponentiation, accelerate CPK based on DLP.
- Up to 1000 Mbps AES bulk encryption.
- Tamper-proof key storage.
- PKCS #11 API and PCI-E interface.
- Support Solaris and Linux.
- Price $1,350

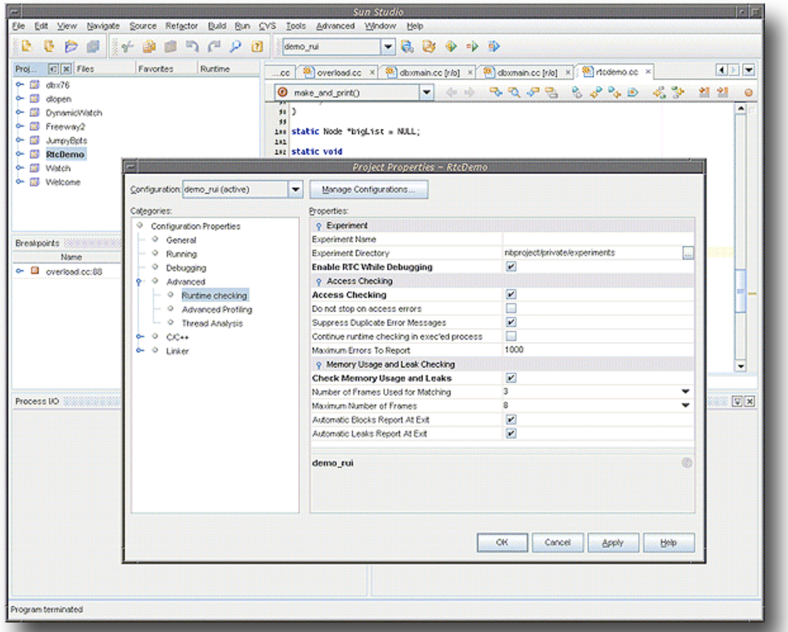
IDE



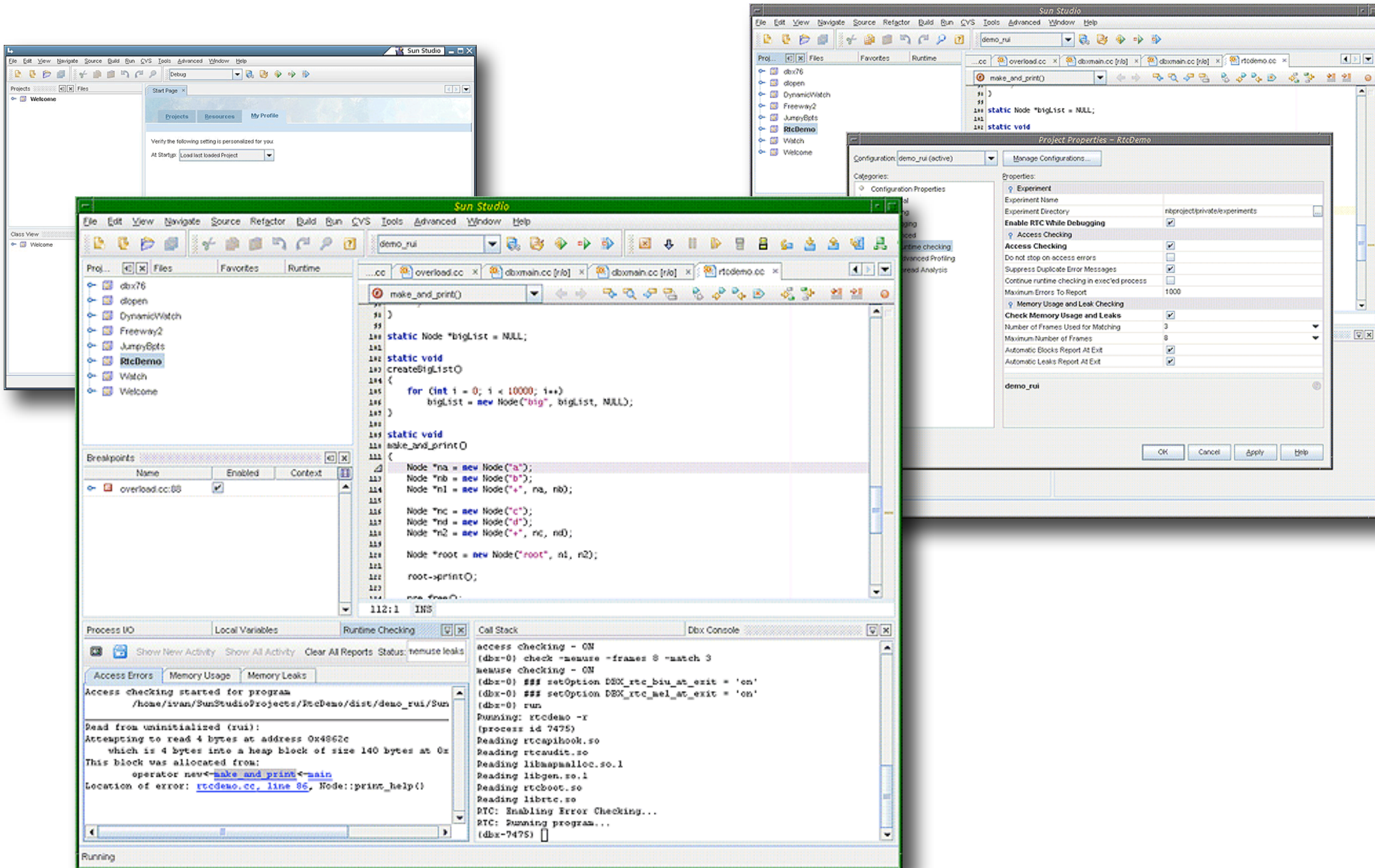
Sun Studio
File Edit View Navigate Source Build Run CVS Tools Advanced Window Help
Debug
Projects
Files
Welcome
Start Page
Projects
Resources
My Profile
Verify the following setting is personalized for you:
At Startup:
Load last loaded Project
Class View
Welcome

IDE

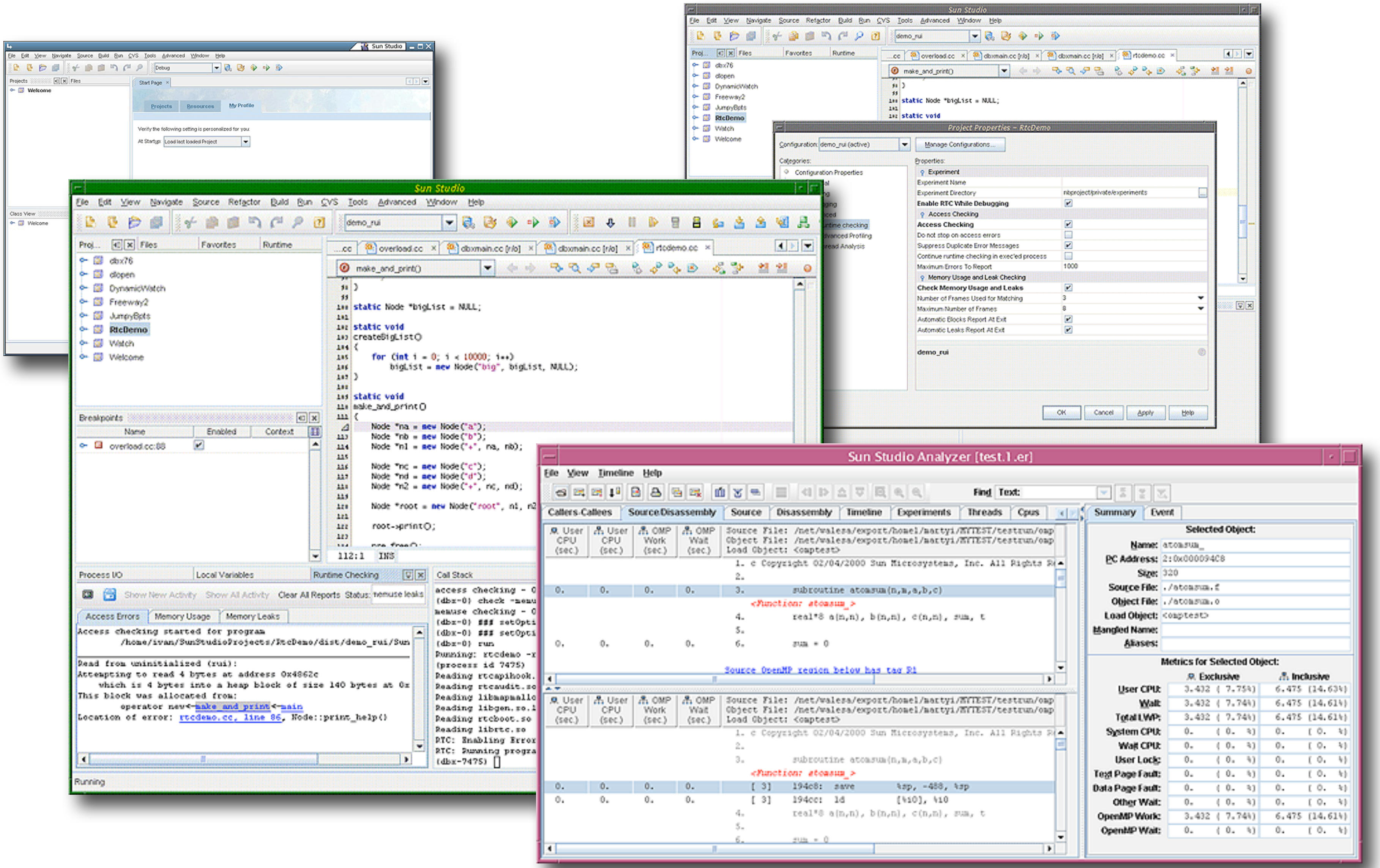

Sun Studio
Start Page
Welcome


Sun Studio
Project Properties – RtcDemo
Configuration Properties
General
Running
Debugging
Advanced
Runtime checking
Advanced Profiling
Thread Analysis
Linker
Experiment
Access Checking
Memory Usage and Leak Checking
OK
Cancel
Apply
Help
Program terminated

IDE

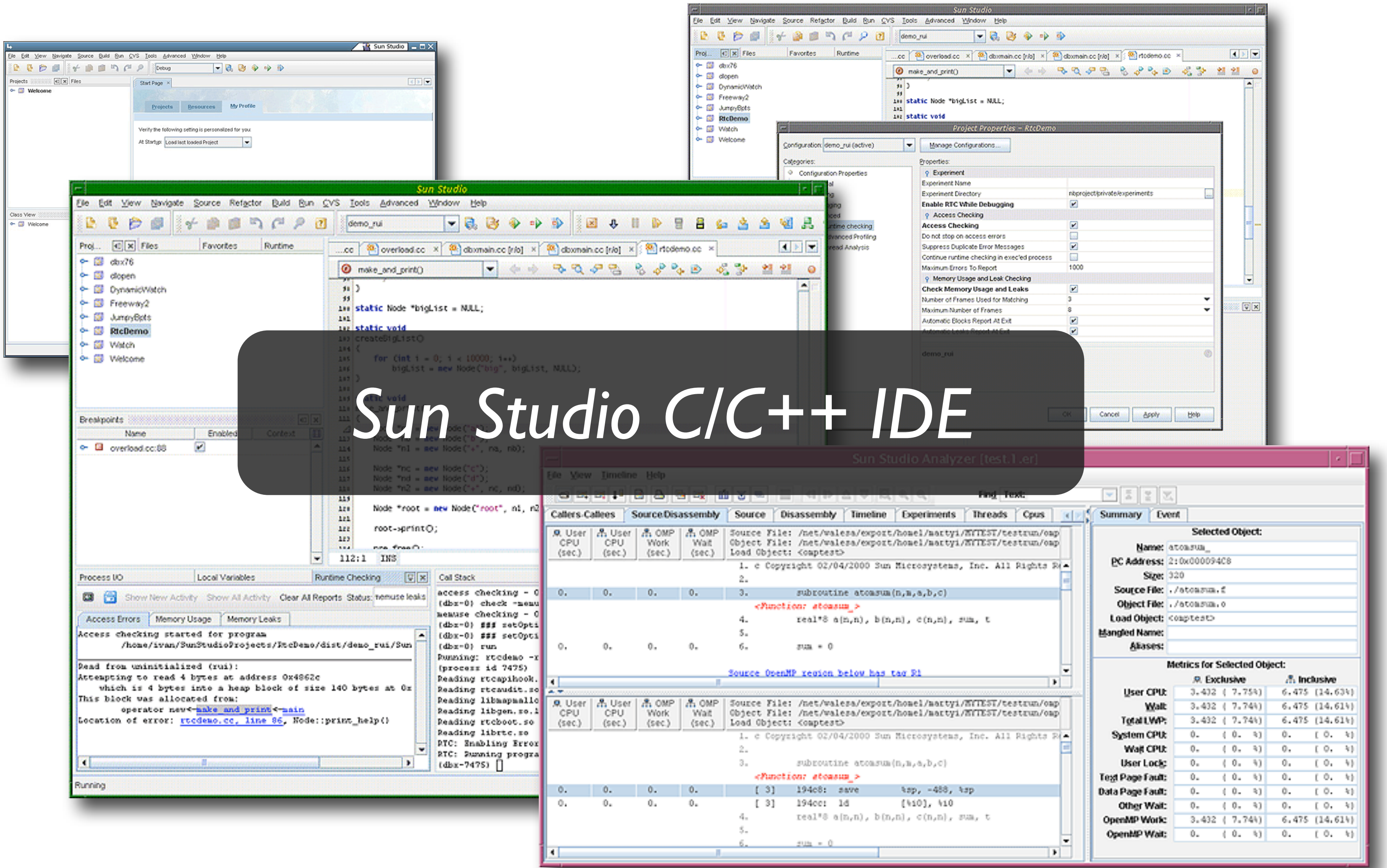

Sun Studio
File Edit View Navigate Source Refactor Build Run CVS Tools Advanced Window Help
Project Properties - RtcDemo

IDE



Sun Studio
Sun Studio
Project Properties - RtcDemo
Sun Studio Analyzer [test.1.er]

IDE



Thanks!

Any Questions?