

# open



USE



IMPROVE



EVANGELIZE

## OpenSSL PKCS#11 engine

Vladimir Kotal  
Security sustaining engineer  
Sun Microsystems Inc.

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
:~::~  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை



# The concept of OpenSSL ENGINES

- An engine is a module pluggable into OpenSSL
  - EVP\* + {RSA,DSA,DH} interfaces **only**
  - ♦ low level interfaces (i.e. `AES_cbc_encrypt()`) do not work with engines
    - An engine is basically a subset of functions which provide own implementation and bypass OpenSSL native implementation
    - An engine does not have to “cover” all ciphers/operations offered by OpenSSL API
  - ♦ if a cipher is not offered by the engine, native OpenSSL implementation will be used

\*EVP = cryptography based EnVeloPe, see `evp(3openssl)` man page for details



## Solaris specifics

- Currently only one engine: PKCS#11 engine
  - Use `openssl engine -vvv -tt -c` to see OpenSSL engine stack
- No dynamic engines
  - Can't do in Solaris: [crypto-with-a-hole](#) problem
  - US Government Export requirement states that all crypto modules are signed and verification is done upon loading
    - Disabling of the dynamic engines was done by the initial OpenSSL integration covered by PSARC/2003/500 (OpenSSL in `/usr/sfw`)



# OpenSSL PKCS#11 engine

- Bridge between OpenSSL API and Solaris Crypto Framework API (they **don't** know about each other !)
  - Main reason: access to HW crypto
  - Side effect: bug in the engine **or** SCF (multiple layers) **or** driver **or** HW means application instability/errors
- The only OS which ships OpenSSL PKCS#11 engine seems to be Solaris (AFAIK)
- Considered as separate “product”
  - However, bugs are filed under `solaris/solaris-crypto/openssl`
- Part of `libcrypto.so`



# PKCS#11 engine in Solaris

- Solaris out-of-the-box consumers:
  - SunSSH client **and** server (see the `UseOpenSSLEngine` configuration directive in `ssh{,d}_config(4)`)
- ♦ engine is used only for RSA/DSA/symmetric\* ciphers
- ♦ not possible to set engine just for specific cipher
  - Apache (`SSLCryptoDevice` in `ssl.conf`)
- ♦ once enabled, PKCS#11 engine is used for every EVP/RSA/DSA/DH operation
- ♦ not possible to set engine just for specific cipher
- Watch out: Java is **not** using OpenSSL and hence is **not** using OpenSSL PKCS#11 engine
  - rather, it goes to PKCS#11 libraries directly

\*the fact it enables the engine for the ciphers does not mean the processing will always end up in PKCS#11 libraries



## The history at a glance

- It all started with proprietary OpenSSL libraries shipped with Sun Crypto Accelerator 1k/4k drivers
  - Need to make it more generic
- PKCS#11 engine design discussions
- Integrated to ON (pre Solaris 10 FCS) with OpenSSL
  - PSARC/2003/500 OpenSSL in /usr/sfw
- Stabilized around late 2008



## Really short PKCS#11 intro

- PKCS = Public Key Cryptography Standards
  - set of standards published by RSA labs (now EMC)
- Defines API for accessing HW crypto modules
  - HW accelerators, token cards, ...
- Userland part of Solaris Crypto Framework (SCF) implements this API\*
  - via `libpkcs11(3LIB)` (`pkcs11_softtoken(5)`, `pkcs11_kernel(5)`)



## PKCS#11 stack

- PKCS#11 stack can be observed via `cryptoadm(1M)`
  - Good for observing potential underlying problems
  - mechanism info: `cryptoadm list -mv`
  - policy info: `cryptoadm list -p`



# PKCS#11 terminology

- Crypto provider is represented by a *slot*
- A slot offers a set of *mechanisms* (crypto operations)
  - SCF implements special slot called *metaslot*
- wrapper for other slots
- Mechanisms work with *objects*
  - object is normally a key
  - persistent objects are called *tokens*
- To perform an operation one has to open a *session*
  - session is opened against specific slot

# Using the engine from command line

- With openssl(1)

- These (normally, see below) go through the engine:

```
openssl speed -evp aes-128-cbc -engine pkcs11
openssl speed rsa1024 -engine pkcs11
openssl dsaparam -engine pkcs11 -genkey \
    -out dsa_priv
```

- These do **NOT**:

- no EVP interface

```
openssl speed aes-128-cbc -engine pkcs11
```

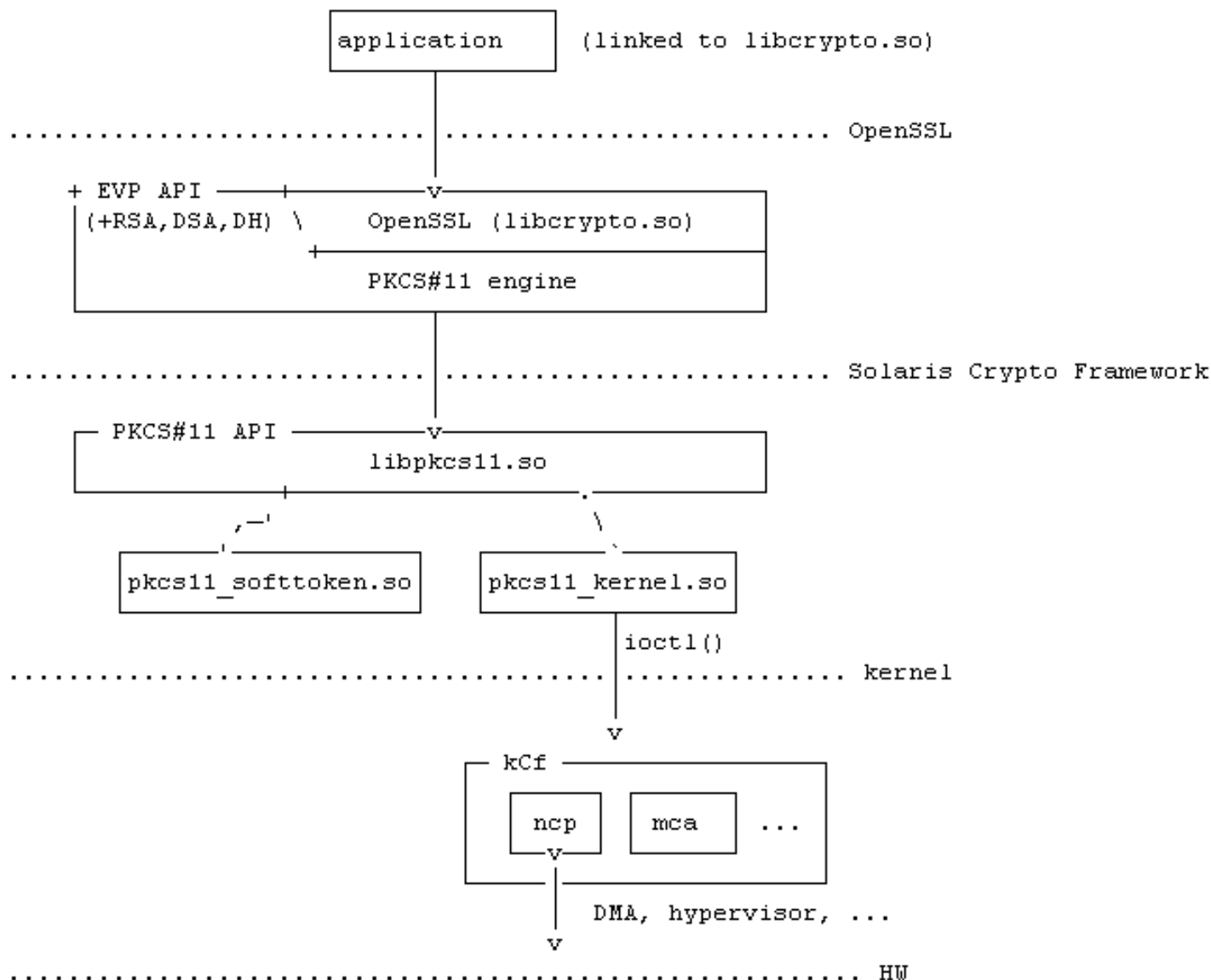
- no engine specification

```
openssl speed rsa1024
```

- this too will not go into the engine **if** there is no HW crypto provider

```
openssl speed -evp aes-128-cbc -engine pkcs11
```

# Layered design





# Use the engine based on workload

- Layering adds non-trivial overhead
- Example: system with `n2cp`
  - HW crypto for symmetric ciphers, e.g. Sun SPARC Enterprise T5120 system
  - It does not make sense to use the engine for small data chunks:

```
$ openssl speed -evp aes-128-cbc -engine pkcs11
```

```
...
```

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes-128-cbc	2180.54k	8662.61k	34622.84k	133167.88k	1096988.72k

```
$ openssl speed -evp aes-128-cbc
```

```
...
```

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes-128-cbc	12211.12k	14629.74k	15640.06k	15934.46k	14071.13k

## How to use the engine (selectively)

- PKCS#11 engine has to be explicitly enabled
  - Multiple reasons:
    - Consumers can make assumptions about EVP OpenSSL API which changes when the engine is enabled
    - fork(2) behavior (unaware consumers can fork which is not the right thing to do in terms of PKCS#11 spec)
    - Different performance characteristics of various operations for given workload (also think layering which adds overhead)
- Possible to select which ciphers will be offloaded
  - `ENGINE_set_default*()`
  - group of specific functions + 1 generic, see `engine(3openssl)`

# Example #1 (selective init/finish)

- see how SunSSH initializes/finishes the engine

- usr/src/cmd/ssh/libssh/common/engine.c

- selective initialization of the engine for DH only:

```
void init_engine(void) {
ENGINE_load_builtin_engines();
if (!(e = ENGINE_by_id("pkcs11")))
errx(1, "cannot load PKCS#11 engine");
if (!ENGINE_init(e)) { /* 'e' is global variable */
ENGINE_free(e); errx(1, "cannot initialize PKCS#11
engine");
}
if (!ENGINE_set_default_DH(e)) /* use the engine only
for DH */
errx(1, "cannot enable engine for DH: %s",
ERR_error_string(ERR_get_error(), NULL));
}
void finish_engine(void) {
/* 'e' is global variable */
ENGINE_finish(e); /* counterpart to ENGINE_init() */
ENGINE_free(e); /* counterpart to ENGINE_by_id() */
}
```



## Example #1 (selective use)

- Except engine initialization/finish nothing changes:
  - it's just bracketing, that's all

```
ENGINE *e = NULL;
int main {
    init_engine_DH(); /* might exit for us */

    if ((dh = DH_new()) == NULL) /* normal OpenSSL API call */
        errx(1, "failed to allocate memory for DH key");

    /* set the modulus (dh->p) and generator (dh->g) here */

    if (DH_generate_key(dh) == 0) /* normal OpenSSL API call */
        errx(1, "failed to generate DH key through PKCS#11 engine");

    finish_engine(); /* might exit for us */
}
```



## PKCS#11 engine sources

- Used to be part of OpenSSL src tree in ONNV gate
- Moved to SFW gate (since snv\_117)
  - 6840551 Move OpenSSL from ON to SFW - SFW
- onnv gate (previously)
  - `usr/src/common/openssl/crypto/engine/\hw_pk11*.[ch]` (7140 lines as of snv\_116)
  - Changes to `eng_all.c`, `eng_cnf.c`, `engine.h` (PKCS#11 engine glue)
  - Note: the code still lives there but will be removed soon
- sfwnv gate (current)
  - `usr/src/lib/openssl/Patches/15-pkcs11_engine-0.9.8a.patch`



# Source code breakdown

- `hw_pk11.c`
  - EVP\_CIPHER definitions, initialization/finish functions (the glue), atfork handlers, session manipulation functions, symmetric cipher API, digest API, RNG API, asymmetric API (only destroy functions !), slot selection functions\*
- `hw_pk11_pub.c`
  - asymmetric API (RSA, DSA, DH), active list manipulation
- `hw_pk11_err.c`
  - error strings, error functions
- `hw_pk11_err.h`
  - function/retval error codes prototypes, function declarations, data structure definitions (PK11\_SESSION)



## Engine awareness

- The app has to explicitly enable the engine
  - recompilation needed to add engine support
  - possible “solution”: add environment variable
- But, after enabling the engine, everything is transparent
  - e.g. `RSA_public_encrypt()` (part of public OpenSSL API) internally “calls” a function pointer which normally points to native OpenSSL implementation (`RSA_eay_public_encrypt()`)
  - ♦ after the engine is enabled the pointer is overwritten with address of `pk11_RSA_public_encrypt()` (which calls `pk11_RSA_public_encrypt_low()` which calls `C_encrypt()`)



# Types of interposing

- Pointer overwrite
  - e.g. the `RSA_public_encrypt()` case mentioned earlier
- Call replacement
  - engine functions have the same prototypes
  - e.g. call to `RSA_sign(..., RSA *rsa)` is replaced by a call to `pk11_RSA_sign(..., RSA *rsa)*`
- As a result all calls to engine-ready OpenSSL public API pass the pointer to OpenSSL data structures



## Design

- As a bridge between PKCS#11 API and OpenSSL API, the engine has to track the data from both worlds
  - PKCS#11 operates with *handles*\*
  - each object (think a key) is represented by *object handle*
  - operations with objects are done in *sessions*, represented by *session handles* (a session is opened against PKCS#11 *slot* which represents PKCS#11 *provider* such as HW crypto unit)
  - the same session can be used for different operations (typically in sequential order)
  - an app can have multiple sessions opened against various slots
- At the same time, it should add as little overhead as possible
  - the fate of being a translator/middle layer

\*an opaque pointer



# Achieving minimal overhead

- Main goal: avoid `C_Find*()` operations for key lookup
  - `C_Find*()` ops lock up the slot and they are expensive (bignum comparisons)
  - Solution: cache PKCS#11 object handles together with pointers to OpenSSL key structures in a special structure
- ♦ This is the `PK11_SESSION` structure, internal to the engine



## PK11\_SESSION structure

- This structure is basically a *cache entry*
  - each structure has a PKCS#11 session handle
  - it also contains object handle and OpenSSL structure pointer
  - pairs PKCS#11 object handle(s) with pointers to OpenSSL structures (RSA, DSA, DH) or raw data (symmetric key)
  - not needed for digest/random ops (they only use the session handle)
- Each operation type has its own cache of PK11\_SESSION structures
  - the cache is just a LIFO
  - `pk11_get_session()` removes the first entry in the list and returns it, `pk11_return_session()` places the structure to the head of the list (without making any change to it)



# Different uses of cache entries

- **Stateful/stateless operations**
  - digest/symmetric crypto functions do not return the PK11\_SESSION structure back to cache until done
  - The state is stored in the CTX structure (see e.g. `hw_pk11.c:pk11_cipher_do_cipher()`).
    - random/asymmetric crypto functions get a session for each operation
- ***Caching/non-caching***
  - Only asymmetric/symmetric functions cache the key data in PK11\_SESSION structure

# Caching functions

- **Caching PKCS#11 engine functions work like this:**
  1. get a cache entry via `pk11_get_session()`
  2. check if the entry is a cache hit or miss via `check_new_*key*()`
    - compares pointers and some components of the key
      - pointer comparison is not always sufficient (CRs 6707274, 6709057)
    - if it is a miss, the object cached in PK11\_SESSION is destroyed\* via `pk11_destroy_*object*()` and reinitialized via `C_Find*()/C_CreateObject` in `pk11_get_*key*()`
  3. perform the operation using the session and object handle (e.g. `C_Sign()`)
  4. return the cache entry back via `pk11_return_session()`

\*not always, will be discussed in reference counting slides in a while



# Cache representation

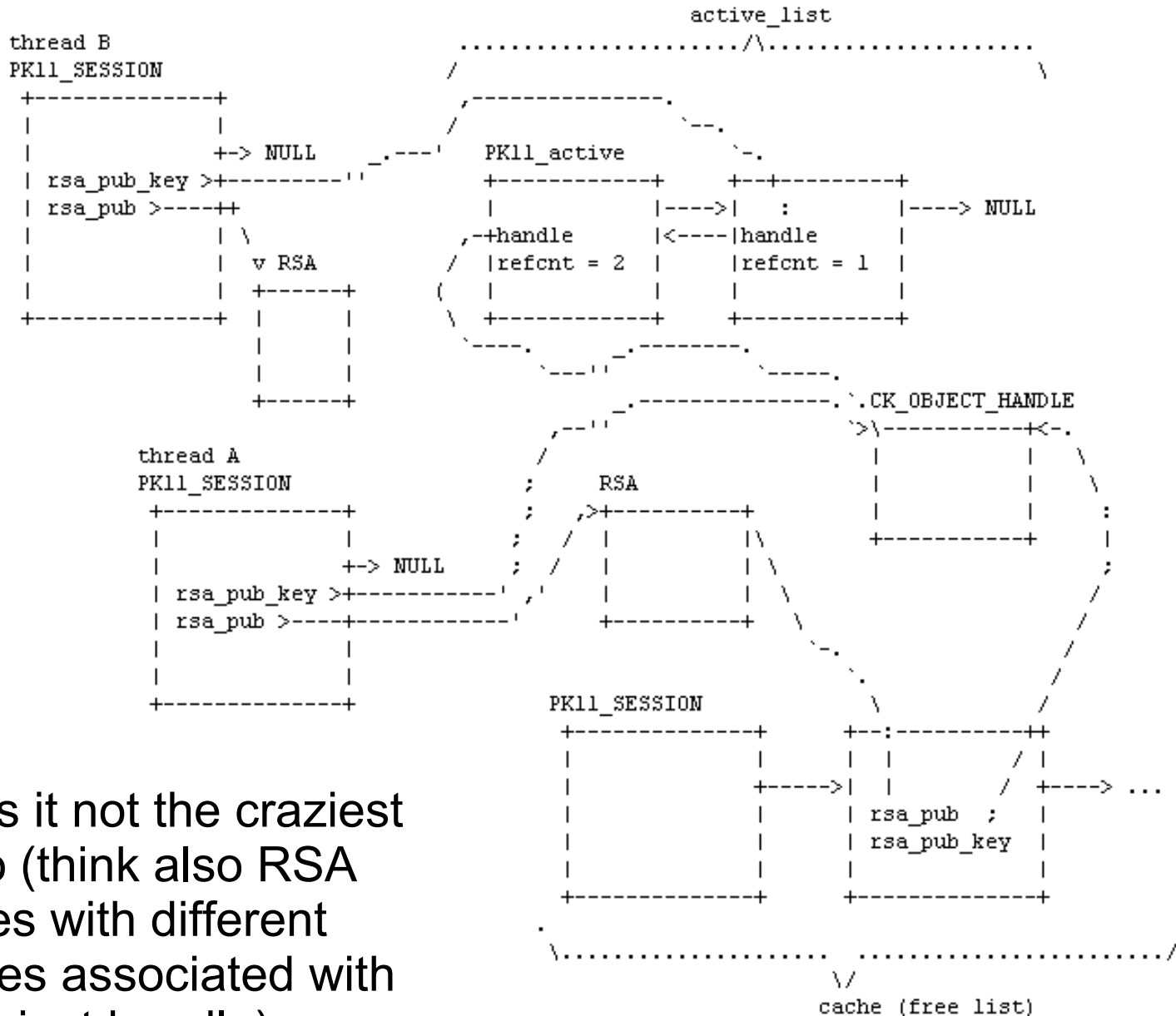
- **PK11\_CACHE structure**
  - list of `PK11_SESSION` structures and its lock
- **Simple linked lists: `freelist`**
  - list of available `PK11_SESSION` structures (the cache)
  - initially it was single list for all operations
  - ♦ later split into per-op-type lists with granular locking (CR 6723504)



## Problem with the cache

- Multiple OpenSSL structures can map to single object handle
  - need to perform reference counting so one thread does not destroy the object while another thread is working with it
- Lists of referenced handles: `active_list`
  - introduced with cache poisoning fix (CR 6602801)
  - for asymmetric operations only
  - double linked list of `PK11_active` structures, each contains object handle referenced by some `PK11_SESSION` structure
- both in `freelist` **or** owned by some thread

# Reference counting (example)



NOTE: This is not the craziest scenario (think also RSA structures with different addresses associated with single object handle)

# Locks in the engine

- Each operation type has its own cache
  - Op types are defined via the `PK11_OPTYPE` enum in `hw_pk11_err.h`
  - The consistency of the cache must be protected
  - `session_cache` is array of `PK11_CACHE` structures (indexed by op type) which contain pointer to the head of the list and a lock
- Groups of `C_Find*()` operations must be atomic
  - PKCS#11 spec v2.20 says in 11.7 Object Management Function, section for `C_FindObjectsInit()`: *“at most one search operation may be active at a given time in a given session”*
  - `find_lock` is array of mutexes, indexed by op type



## Slot selection

- Some operation types can have its own slot
  - asymmetric ops share one slot (`pubkey_SLOTID`), random has its own (`rand_SLOTID`), `symm/digest` share one (`SLOTID`)
  - The slots are determined at engine initialization time
- ♦ `pk11_choose_slots()` opens metaslot and goes through the mechanisms in all slots it offers

# Slot selection for ciphers/digests

- `pkcs11_softtoken`'s implementation of ciphers/digests is slower than OpenSSL native
  - we do not want to use engine for ciphers/digests if there is no HW crypto in the system
  - But: can't use `pkcs11_kernel` directly because OpenSSL native is slower for RSA/DSA/DH than `softtoken`
- With HW crypto it looks like this\*:

```
$ SOFTTOKEN_DIR=/tmp truss -ulibcrypto::pk11_* \
    openssl speed -evp aes-128-cbc -engine pkcs11
...
/1@1:  -> libcrypto:pk11_cipher_do_cipher(0xffbfd924, 0x882c8, 0x882c8,
0x10)
/1:    ioctl(3, CRYPTO_ENCRYPT_UPDATE, 0xFFBFD26C)      = 0
/1@1:  <- libcrypto:pk11_cipher_do_cipher() = 1
```

\*without HW crypto you won't see a thing called in `pk11_cipher_*`() namespace



# Cipher/digest selection in detail

- `pk11_library_init()`
  1. `check_hw_mechanisms()` is called first
- `dlopen()`'s `pkcs11_kernel.so` and populates 2 lists of mechanisms (`hw_cnids` for symm. ciphers, `hw_dnids` for digests\*) it finds there (all HW slots)
- 2. `pk11_choose_slots()` checks HW supported mechs found in first step against mechs found in metaslot
- `pk11_get_symmetric_cipher()/pk11_get_digest()` are the who decide (they put NIDs into local tables)
- as a result `{cipher,digest}_nids*` tables are only populated with NIDs offered by HW
  - `bind_pk11()` finishes the job via `ENGINE_set_{ciphers,digests}()` and `pk11_engine_{ciphers,digests}()`

\*global variables



## Slot selection could be better

- Slot selection for asymm. ops based on maximum supported key length and combination of algorithms offered (CR 6755119)
- **Define** `DEBUG_SLOT_SELECTION` macro when **compiling** `libcrypto.so` **to see more**
  - should be easier after CR 6731376 is fixed (env variable)



## PKCS#11 engine and fork(2)

- PKCS#11 spec (v2.20) warns in section 6.6.1 Applications and processes:
  - *“In this particular case (when C is the child of a process which is a Cryptoki\* application), the behavior of Cryptoki is undefined if C tries to use it without its own C\_Initialize call.”*
- Why is this a problem ?
  - in-kernel state (session closed in parent affects the child)

\*In PKCS#11 spec, Cryptoki (cryptographic token interface) is the name of the API



## Working around fork()

- Most applications do not care
- The engine contains soft pillow for apps which use it incorrectly in terms of forking
  - fork() is detected by comparing the `pid` member of `PK11_SESSION` in `pk11_get_session()` with value returned by `getpid()`. In such case everything in the engine is free'd and reinitialized.
- Might be possible to handle this also by saving the state
  - use `C_{Get,Set}OperationState()`
  - further work needed on Solaris Crypto Framework (CR 6636963 for digests)



# What an app should do when forking

- Enable, use, disable, fork, enable, etc.
- SunSSH uses the engine correctly
  - see `process_newkeys()` in `usr/src/cmd/ssh/libssh/common/packet.c`
- `mod_ssl` used by Apache only initializes the engine
  - in `pkg.sslmod/ssl_engine_init.c:ssl_init_Engine()`
  - also, it does not bother to call `ENGINE_finish()`
  - should finish and re-init in `ssl_init_Child()`
- ♦ we should work with Apache to get this fixed (PKCS#11 engine specific code)\*



## Example #2 (correct use w.r.t. fork())

```
init_engine();
/* do some work */
finish_engine();
switch(fork()) {
    case 0: /* child */
        init_engine();
        /* enter client specific function */
        finish_engine();
        exit(0);
    case -1:
        err(1, "fork failed");
    default: /* parent */
        break; /* FALLTHRU */
}
/* parent continues here */
init_engine();
/* parent does its job here */
finish_engine();
```



## atfork handlers

- The engine is part of a library and has its own locks after 6723504 (granular locking)
  - Necessary to do the right thing in threaded environment (deadlock avoidance)
- `pk11_fork_{prepare,parent,child}()` **handlers**
  - prepare acquires all locks in the engine (pre-fork), {parent,child} release the locks
  - should be possible to detect fork(2) using the handlers (RFE 6854930)

## AES CTR mode

- Problem: OpenSSL EVP API does not support AES CTR modes
  - but we still want to use it (esp. in SunSSH)
- Added with CR 6685012 (new cipher modes)
- Solution: supply our own NIDs
  - global variables: `NID_aes_{128,192,256}_ctr`
  - Dynamically set them during engine initialization
    - ♦ via `pk11_add_aes_ctr_NIDs()`
    - ♦ add special cases into `pk11_init_symmetric()` and `pk11_engine_ciphers()`
- Users have to fill the `EVP_CIPHER` structure
  - see `usr/src/cmd/ssh/libssh/common/cipher-ctr.c` for example<sub>38</sub>



# Debugging

- `truss(1)` helps with basic debugging
  - `-ulibcrypto:: -ulibpkcs11::`  
`-upkcs11_softtoken:: -upkcs11_kernel`
  - Use `-ulibcrypto::pk11_*` to see just engine calls
  - useful for watching PKCS#11 CKR values (return/error codes)
- Cache observability
  - `check_new*_key()` returns 1 on cache hit, 0 on miss



# Debugging multithreaded issues

- Dtrace needed to debug multithreaded programs
  - need to copyout() the structures
  - see internal data (i.e. `magic_marker` in `soft_session_t`)



# Resources

- Jan Pechanec's blog
  - <http://blogs.sun.com/janp/>
- PKCS#11 engine patch against upstream
  - [http://openssl.org/contrib/pkcs11\\_engine-\\*.tar.gz](http://openssl.org/contrib/pkcs11_engine-*.tar.gz)
  - can be used e.g. for Linux
- `engine(3openssl)` man page
- PKCS#11 spec
  - <http://www.rsa.com/rsalabs/node.asp?id=2133>
- OpenSolaris/public mailing list
  - [crypto-discuss@opensolaris.org](mailto:crypto-discuss@opensolaris.org)



## Future work (not set in stone)

- Optimal caching (6669630) and better cache search (6666668)
  - Possible problem: cache explosion (no limit)
- Configurable engine (6591009, 6627948)
  - Turn on/off the engine per cipher
  - enable/disable the cache per cipher type
- Keys by reference (6479874)
  - Ability to store private keys in HW tokens while accessing them semi-transparently via OpenSSL API
- Integrate PKCS#11 engine to upstream

# open



USE



IMPROVE



EVANGELIZE

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
•••••  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
ОТКРЫТЫЙ  
வெளிப்படை

Vladimir Kotal  
Security sustaining engineer  
Vladimir.Kotal@Sun.COM  
<http://blogs.sun.com/vlad/>

“open” artwork and icons by chandan:  
<http://blogs.sun.com/chandan>