LibTom Projects

LibreSoftwareMeeting
Dijon, France
July 5-9, 2005

Tom St Denis
tomstdenis@gmail.com

Sponsored by

Elliptic Semiconductor Inc.
ellipticsemi.com

Secure Science Corporation
securescience.net

Overview

- Introduction to LibTom projects
- History of projects
- Progressions
- Lessons learned
- Designing a library
- Descriptors
- Portable Software
- Profile Driven Optimization
- Secure Coding

What are the LibTom Projects?

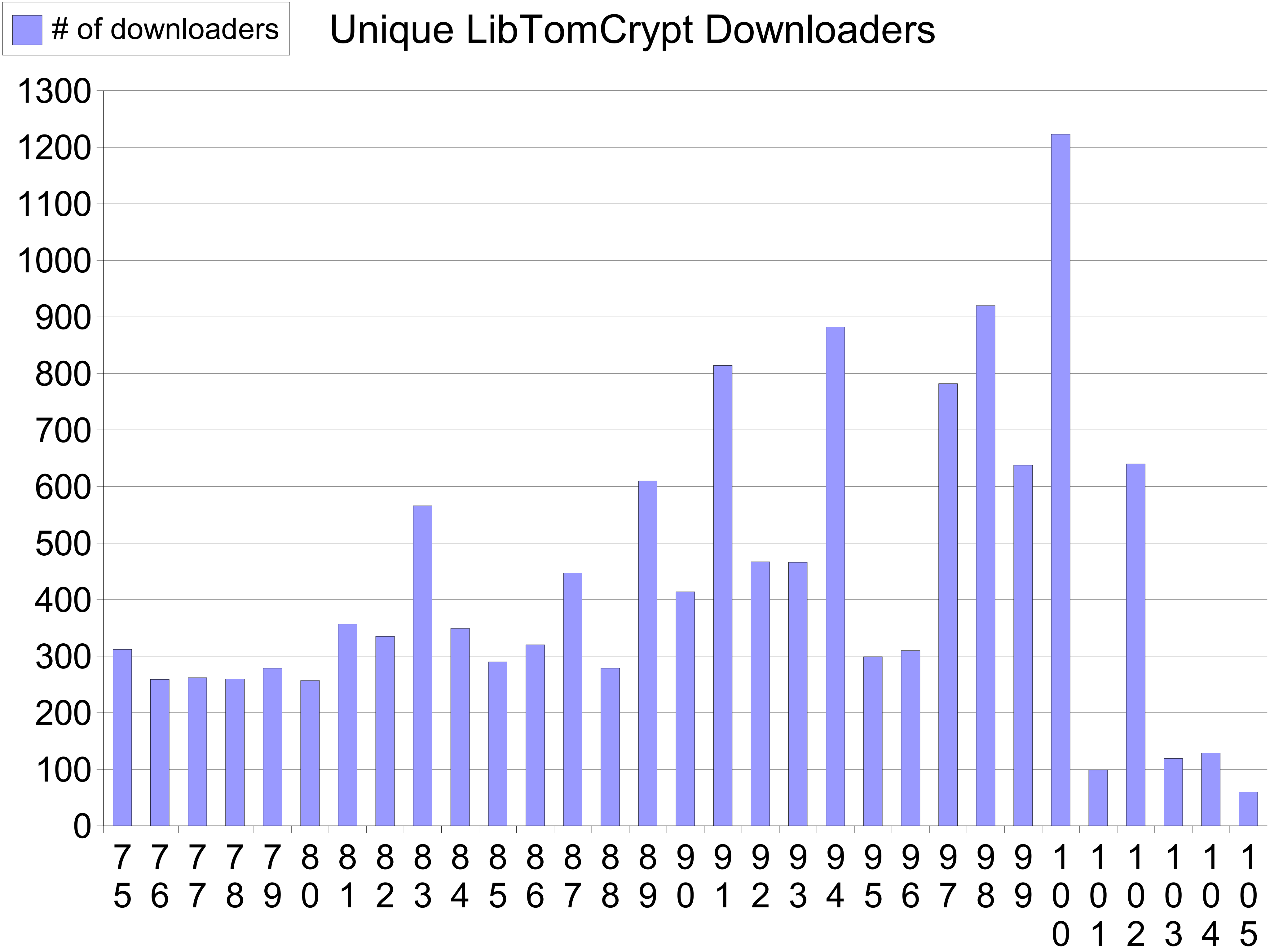
- Series of seven libraries covering cryptography, bignum math, polynomial math, bigfloat math and network security
- All written from scratch in portable C
- Written to work well and be educational
- All public domain
  - Distributed in source form only
  - Manuals are in source + PDF format

Who uses them?

- Industry
  - Products such as video games, embedded devices, network routers, license engines
- Education
  - LibTomMath used as reference
    - Textbook used
  - LibTomCrypt used as toolkit
    - Has been cited academically

Who uses them (2)

- OSS
  - Mozilla NSS, MatrixSSL, Paketto, Torque, Dropbear, TCL, Ruby, Python, SILC, DigSig, Agent++, Sharewidth, FreeOTFE, etc...
- Various
  - Each release gets roughly 500 unique IP downloads
    - Ranging from individuals to corporations
  - Mirrored through BSD ports and Gentoo portage


Unique LibTomCrypt Downloaders
# of downloaders
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105

Why use them?

- Not just free, public domain
  - License portrays values and intentions
- Builds anywhere
  - Very portable (no configuration either)
    - Builds out of box on x86, mips, arm, ppc, ...
- Competitively efficient
- Well documented
  - In total there are over 500 pages of documentation

LibTomCrypt

- Cryptographic toolkit written in C
- Provides
  - ciphers, hashes, PRNGs
  - MACs (and ENC+AUTH modes like CCM/GCM)
  - public key cryptography (RSA PKCS #1, DSA and EC-DSA)
  - ASN.1 DER support
  - simple and consistent API
  - tons of documentation and commented source code
    - LTC: Includes doxygen comments
- Builds out of the box with GCC and MSVC

LibTomMath

- Multi-Precision Math Toolkit written in C
- Provides
  - Basic math (add, sub, mult, div, etc)
  - Optimized routines (mult, reduction, exptmod)
  - Number theoretic functions
  - 300+ page textbook on the subject
  - Well documented and commented source
  - The de facto standard for
    - SILC
    - TCL Scripting Language
    - Ruby Scripting Language
- Builds out of the box with GCC and MSVC

TomsFastMath

- Fast Fixed-Precision Math in C
  - Meant for very fast mult/sqr/exptmod
  - Largely based on LibTomMath
- Provides
  - Basic math (add, sub, mult, div)
  - Easy to tune ASM optimized multipliers
  - Very fast exptmod (faster than OpenSSL)
- Builds with GCC on x86_32, x86_64, ARM and PPC32 boxes
- Mozilla NSS has plans to move to TomsFastMath
  - They rewrote my Montgomery code for me :-)

History of the Projects

- Started LibTomCrypt in Dec'01
  - Winter break from college
  - Written to provide a generic crypto API
  - Almost bought in 2003 by Sony :-)
    - For less than the average cost of a low-priced car
  - 1st release was December 21st 2001
  - 100th release was December 31st 2004
  - Currently version 1.05 (105th release)

History of Projects (2)

- Started LibTomMath in Dec'02
  - ... winter break from college
  - Written to improve upon MPI
    - Faster and easier to read source code
    - Very instructional personally
  - Took only a few months to get stable and competitive

History of the Projects (3)

- Started LibTomNet in Jul'03
  - Summer break ;-)
  - An exercise in networking
- Started LibTomPoly in ... Dec'03
  - ... guess
- Started LibTomFloat in May'04
  - ... Summer break again

Progressions

- Started by using “batch files” as make scripts
  - Quickly replaced that with makefiles
- Makefiles were very “hard coded”
  - Slowly replaced that with a flexible build system
- Target compiler moved from Borland to GCC
  - Used DJGPP and then Cygwin
  - Now use GNU/Linux exclusively
    - Develop on a dual-core AMD64, Intel Prescott P4 and an AMD Athlon XP-M laptop :-)

Progressions (2)

- Used to densely pack code
  - Now few functions (usually just one) per file
  - Finding functions and distributing work easier
    - Also easier to audit and fix
  - LTC 1.05 is 209 lines/file (mpi.c accounting for 9000 lines)
    - 175 lines/file (discounting mpi.c)
- Re-factored the header files
  - Sorted by class
- Code used to be in one directory
  - Sorted by class

Progressions (3)

- Used to do releases daily
  - Take my time, test builds, try configurations
- Always took input from others
  - Actively seek it now though
- Make changes that “suited me”
  - Think about the user “customer” impact

Various Lessons Learned

- Setup is more important than execution
  - Makes deployment easier
  - Makes testing easier
- Intuition is very handy
  - Recent ASN.1 DER bugs
  - Being apprehensive means your cautious

Various Lessons Learned (2)

- A CVS (or SCM) even “locally” is very handy
  - Multi-box development
  - Ability to revert, see differences, etc
  - Simple way to backup too (+cronjob)
  - Prove lineage of code
- Be consistent
  - Similar prototypes, documentation
  - Helps lower learning curve
  - Helps debug as well

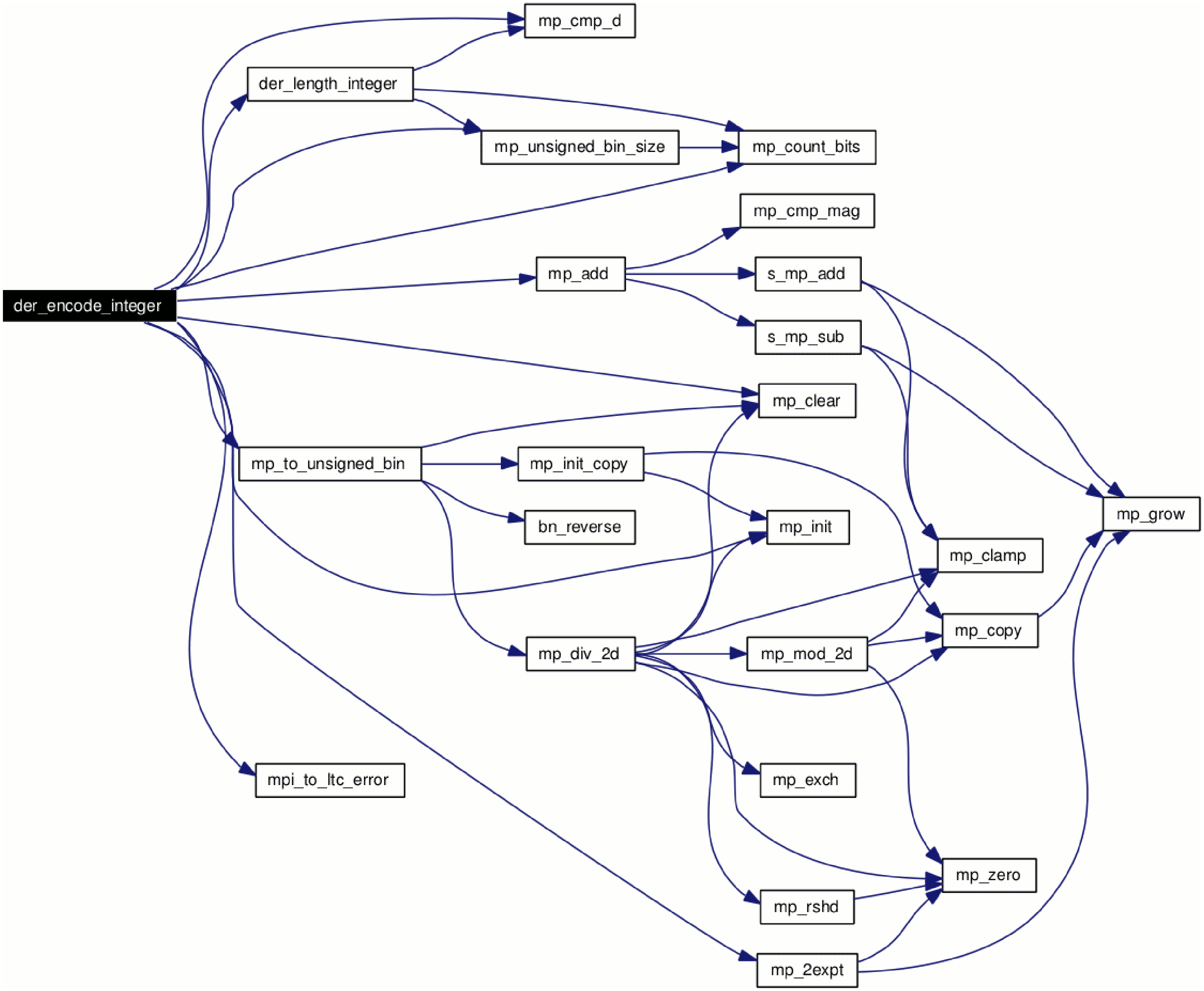
Various Lessons Learned (3)

- Take time between releases
  - “release early, release often” confusing
    - Not really used in practice either
    - All about how you stage a project (setting goals)
  - Tracking multiple releases is more work
    - Support emails can get tricky
    - People **will** use outdated software
  - Spend more time working on product not release
    - Releases take roughly a day to finalize (**after** testing)
  - If effort required not that high do it anyways
    - Looks better and means less to add to a TODO list

Designing a Library

- Identify a clear problem and solution set
  - Find reasonable solutions to given problem
  - Sort solutions by work, benefit ratios
- Start from ground up
  - Design a hierarchy of the functions required
    - Helps profile code as well

Hierarchy.



mp_cmp_d
der_length_integer
mp_unsigned_bin_size
mp_count_bits
mp_cmp_mag
mp_add
s_mp_add
der_encode_integer
s_mp_sub
mp_clear
mp_to_unsigned_bin
mp_init_copy
mp_grow
bn_reverse
mp_init
mp_clamp
mp_copy
mp_div_2d
mp_mod_2d
mpi_to_ltc_error
mp_exch
mp_zero
mp_rshd
mp_2expt

Designing a Library (2)

- Setup a CVS (or SCM) first
  - Helps keep work organized
- Start development with headers for common data types and lowest level functions, error codes, etc.
  - Test as you write
    - Keep as few “untested functions” in your test path
    - If you keep building on a tested hierarchy it's much easier
- Use tools that will be available to the user
  - Use smart dependent libraries

Designing a Library (3)

- Get a stable make system in place early
  - Get it out of the way (one less thing)
- Think of how to test from the start
  - Include how to compare results (e.g. lengths, status)
- Document as you code
  - Don't leave till the end (lesson learned)
- Set reasonable goals and deadlines
  - Have something to keep yourself on track
  - Keep your users informed

Designing a Library (4)

- Avoid “because it's free” as excuse to compromise
  - Lowers the usefulness
  - More work to fix later
  - Creates “more noise”
  - Sets bad impression forth
    - Personal, professional and “OSS” specific
- Support is important
  - Helps users, more users => more eyes
  - Have an “email” address and not just mail list/forums

Promotion

- Website
  - Simple but accurate, don't shroud things
    - What, Why, How and Where
  - Make contact info easy to find
- “Plug'ing”
  - Keep it short and respectful
  - Don't try to hide the plug
  - Use some restraint, have modesty (pays off)
- Word of mouth
  - If you build it ... they will spread it ;-)

The LibTomCrypt Difference “Descriptors”

- **Problem**: common and consistent interface to any number of hashes, ciphers or PRNGs
- **Solution**: table driven “descriptors”
  - A “struct” with pointers to functions
  - Have other data (block size, name) that describes the module
- Similar to a C++ “class” but without the overhead

“Descriptors” (2)

- One type of descriptor per class of function
  - One for ciphers, hashes and PRNGs
- Dependent code only uses tables
  - e.g. CTR mode, OMAC, HMAC, etc
  - Makes them algorithm agnostic
- Makes it easy to switch from one to another
  - e.g. from AES to Twofish
- Also makes using optimized/hardware algo easier

CAST5 Descriptor

```
const struct ltc_cipher_descriptor cast5_desc = {
   "cast5",                              <= Name
   15,
   5, 16, 8, 16,                         <= Parameters (keysizes, block size, rounds)
   &cast5_setup,                         <= Pointers to the functions
   &cast5_ecb_encrypt,
   &cast5_ecb_decrypt,
   &cast5_test,
   &cast5_done,
   &cast5_keysize,
   NULL, NULL, NULL, NULL, NULL, NULL, NULL
};
```

Various Snippets

In **ctr_encrypt.c**

```
      /* encrypt it */
      cipher_descriptor[ctr->cipher].ecb_encrypt(ctr->ctr, ctr->pad, &ctr->key);
      ctr->padlen = 0;
```

In **omac_process.c**

```
   /* ok if the block is full we xor in prev, encrypt and replace prev */
   if (omac->buflen == omac->blklen) {
      for (x = 0; x < omac->blklen; x++) {
         omac->block[x] ^= omac->prev[x];
      }
      cipher_descriptor[omac->cipher_idx].ecb_encrypt(
         omac->block, omac->prev, &omac->key);
```

“Descriptors” (3)

- Also makes interfacing with optimized modules (e.g. hardware) easy
  - Users can link against third party descriptors without rebuilding LibTomCrypt
- Overall very flexible code base
  - Has been contorted quite a bit in the field

“Descriptors” (4)

- Recent changes
  - Cipher descriptors lacked support for hardware
    - No “multi-block” modes
    - No “done” function
  - Updating code base was trivial but time consuming
    - Should have done it originally
    - Not part of the original problem set
      - The problem evolves!

Portable Software

- Definition of “portable”
  - You want to hear “it just works”
- “autoconf” is not your friend
  - In fact some platforms don't have autoconf!
- Target language
  - C99 becoming more prevalent
  - Usually C90 is good enough
    - Target C90 and you're covered
    - Avoid some new C99 features (VLA and “restrict” for instance)
    - “long long” very common at least

Portable Software (2)

- Target Toolchains
  - You can expect to have
    - make, gcc, cc1, as, ld, ar, objcopy, objdump, bash/tcsh/sh
  - More realistically
    - $(CC), $(AR), $(LD), ...
  - You can't expect to have
    - gdb, gprof, libtool, a shell other than “sh”, other compilers, linkers, etc.
    - Not always GNU variants either
      - Think “unix make” vs. “GNU make”

Portable Software (3)

- Target Build
  - Usually static archive
  - Ideally provide the ability to libtool a shared object
  - Avoid excessive flags
    - -O2 or -O3 is enough (actually read the man pages about the combinations)
      - e.g. "-funroll-all-loops" is usually bad for performance
    - Warnings are being added to new GCCs, assume people are using gcc 3.0-3.3 not 3.4
      - But do make them accessible (as they are handy)

Portable Software (4)

- Make no assumption about types
  - Char can be signed
  - Ranges are **minimums**
  - Floating point types
  - Endianess (if in doubt think neutral)
- Careful for
  - Right shifts (undefined for signed)
  - Casts to/from non-void
  - Structure alignment

Portable Software (5)

- Endianess issues
  - Aim for at least neutral
  - Add support for common platforms afterwards
    - mips, x86, ppc, ...
      - Not always possible to optimize neatly
- Inline asm
  - Sometimes required (profile!)
  - Hazardous
    - Can make "configuration nightmare" happen
    - Not all GCCs are equal even on the same platform!
      - # of registers changes based on build flags too

Portable Software (6)

- Other Oddities
  - Not all platforms have a heap
    - Use flexible heap macros instead
  - Not all platforms have a hardware divider
    - Avoid division at all costs
  - Safe bet that stack usage >4KiB is a bust
    - Use heap as trade off
  - Unaligned writes are not part of ISO C
  - Avoid using FPU related instructions if at all possible

Profile Driven Optimization

- Messed up code
  - Usually the accepted “path” for speed
  - Usually not that much faster anyways
- LT approach
  - Lots of error checking
  - Lots of optimization in specific hot spots
  - No diminishing gains optimizations
    - e.g. no point making code twice as hard to read for 1% speed
  - Be one with the compiler output
    - Know how to read assembler

Profile Driven Optimization (2)

- Pick efficient algorithms
  - Comba not baseline (multiplication)
  - Sliding window exponentiation not square-multiply
  - Identify their hotspots
    - Small pockets of “optimization” usually pay off
- Worth is not just in speed
  - Ports of LT projects
  - Educational use
- Experimentation

Profile Driven Optimization (3)

- Compilers are smart
  - GCC 3.4.x in particular
    - Understands “add with carry” and common multiplication code
    - Can perform code re-arrangement (unroll, hoisting, etc)
  - Can usually schedule code very efficiently
- CPUs are smart too
  - Athlon has deep OOE and three pipelines
    - On most code quickly written assembler won't be faster
      - Takes a lot of time to get smaller gains

Comba Multiplier

From bn_fast_s_mp_mul_digs.c

Inner loop ($O(n^2)$ level):

```
   for (iz = 0; iz < iy; ++iz) {
     _W += ((mp_word)*tmpx++)*((mp_word)*tmpy--);
   }
```

Conversion up to “mp_word” required or we get 32x32=32 .

```
.L130:
        movl    (%ebp), %eax
        mull    (%edi)              <= note GCC is doing 32x32=64
        addl    %eax, %ebx          <= 64-bit add using add/adc
        adcl    %edx, %esi
        subl    $4, %ebp
        addl    $4, %edi
        decl    %ecx
```

Note: GCC 3.4.3 can't do the same for 64-bit CPUs! (YMMV)

Profile Driven Optimization (4)

- Results
  - LTC ciphers/hashes comparable to OpenSSL
  - LTM faster than most, about ½ of OpenSSL
    - Beats out the popular LIP, RSAREF and MPI
      - Also easier to read...
  - TomsFastMath faster than or equal to OpenSSL
    - Also easier to read as it's mostly C
      - and based off of LibTomMath

Profile Driven Optimization (5)

- TomsFastMath approach
  - Mostly portable C ripped from LTM
  - ASM macros used in key locations
    - Multiplication
    - Squaring
    - Modular Reduction
  - All macros have the same interface
  - Routines implemented once, only which macros are activated changes at build time
  - Achieves high performance with low code maintenance

TomsFastMath Macros (ISO C)

```
#define MONT_START
#define MONT_FINI
#define LOOP_END
#define LOOP_START mu = c[x] * mp

#define INNERMUL                                       \
   do { fp_word t;                                     \
   _c[0] = t  = ((fp_word)_c[0] + (fp_word)cy) +       \
                (((fp_word)mu) * ((fp_word)*tmpm++));  \
   cy = (t >> DIGIT_BIT);                              \
   } while (0)

#define PROPCARRY \
   do { fp_digit t = _c[0] += cy; cy = (t < cy); } while (0)
```

TomsFastMath Macros (ARMv4)

```
#elif defined(TFM_ARM)
#define MONT_START
#define MONT_FINI
#define LOOP_END
#define LOOP_START    mu = c[x] * mp

#define INNERMUL                    \
asm(                                \
    " LDR    r0,%1            \n\t" \
    " ADDS   r0,r0,%0         \n\t" \
    " MOVCS  %0,#1            \n\t" \
    " MOVCC  %0,#0            \n\t" \
    " UMLAL  r0,%0,%3,%4      \n\t" \
    " STR    r0,%1            \n\t" \
:"=r"(cy),"=m"(_c[0]):"0"(cy),"r"(mu),"r"(*tmpm++),"1"(_c[0]):"r0");

#define PROPCARRY                  \
asm(                               \
    " LDR   r0,%1            \n\t" \
    " ADDS  r0,r0,%0         \n\t" \
    " STR   r0,%1            \n\t" \
    " MOVCS %0,#1            \n\t" \
    " MOVCC %0,#0            \n\t" \
:"=r"(cy),"=m"(_c[0]):"0"(cy),"1"(_c[0]):"r0");
```

TomsFastMath Macros (x86_64)

```
#elif defined(TFM_X86_64)
#define MONT_START
#define MONT_FINI
#define LOOP_END
#define LOOP_START mu = c[x] * mp

#define INNERMUL                                           \
asm(                                                       \
   "movq %5,%%rax \n\t"                                    \
   "mulq %4       \n\t"                                    \
   "addq %1,%%rax \n\t"                                    \
   "adcq $0,%%rdx \n\t"                                    \
   "addq %%rax,%0 \n\t"                                    \
   "adcq $0,%%rdx \n\t"                                    \
   "movq %%rdx,%1 \n\t"                                    \
:"=g"(_c[LO]), "=r"(cy)                                    \
:"0"(_c[LO]), "1"(cy), "r"(mu), "r"(*tmpm++)               \
: "%rax", "%rdx", "%cc")

#define PROPCARRY                            \
asm(                                         \
   "addq   %1,%0    \n\t"                    \
   "setb   %%al     \n\t"                    \
   "movzbq %%al,%1 \n\t"                     \
:"=g"(_c[LO]), "=r"(cy)                      \
:"0"(_c[LO]), "1"(cy)                        \
: "%rax", "%cc")
```

TomsFastMath Macros (PPC32)

```
#elif defined(TFM_PPC32)
#define MONT_START
#define MONT_FINI
#define LOOP_END
#define LOOP_START mu = c[x] * mp

#define INNERMUL                              \
asm(                                          \
   " mullw    r0,%3,%4          \n\t"  \
   " mullhwu  r1,%3,%4          \n\t"  \
   " addc     r0,r0,%0          \n\t"  \
   " addze    r1,r1             \n\t"  \
   " lwz      r2,%1             \n\t"  \
   " addc     r0,r0,r2          \n\t"  \
   " addze    %0,r1             \n\t"  \
   " stw      r0,%1             \n\t"  \
:"=r"(cy),"=m"(_c[0]):"0"(cy),"r"(mu),"r"(*tmpm++),"1"(_c[0]):"r0", "r1", "r2");

#define PROPCARRY                             \
asm(                                          \
   " lwz      r0,%1             \n\t"  \
   " addc     r0,r0,%0          \n\t"  \
   " stw      r0,%1             \n\t"  \
   " xor      %0,%0,%0          \n\t"  \
   " addze    %0,%0             \n\t"  \
:"=r"(cy),"=m"(_c[0]):"0"(cy),"1"(_c[0]):"r0");
```

This is untested code. If you have a PPC box please see me after my talk :-)

Secure Coding

- Trust
  - Don't trust input pointers
    - Check for NULL
  - Don't trust the contents of input structures
    - Check index boundaries
      - Performance loss is negligible
  - Beware of signed vs. unsigned issues
    - “x < y” can fail in meaning if y < 0 and x is unsigned

Typical Bounds Checking

From omac_process.c

Check for NULLs

```
  LTC_ARGCHK(omac   != NULL);            <= like assert macros
  LTC_ARGCHK(in     != NULL);
```

Check array boundaries

```
  if ((err = cipher_is_valid(omac->cipher_idx)) != CRYPT_OK) {
     return err;
  }
  if ((omac->buflen > (int)sizeof(omac->block)) || (omac->buflen < 0) ||
      (omac->blklen > (int)sizeof(omac->block)) || (omac->buflen > omac->blklen)) {
     return CRYPT_INVALID_ARG;
  }
```

Secure Coding (2)

- Resource Checking
  - Keep track of buffer size remaining
    - Avoid overflows and overruns
    - Example, DER encoder scans for sizes before writing a single byte
  - Do this as you code to avoid “lazy programmer” problems
  - Avoid typical bad functions
    - gets, scanf, sprintf, strcpy, strcat, etc...

Secure Coding (3)

- Error checking
  - Check all return codes
    - Avoid “lazy programmer” syndrome
  - Be consistent with error codes and their usages
- Oddities
  - Use calloc not malloc
  - Think about thread safety

About Myself

- Started “coding” at age 12 (1994)
  - Wrote a BBS in Pascal and taught myself C
  - “hacked” Turbo-C lite to allow programs to run outside the IDE shortly afterwards ;-)
- Wrote a crypto messaging program at age 17
  - Almost got sued by RSA for using RC5
- Started LibTomCrypt at age 19
  - Basically have radically changed the way I develop software ever since

Elliptic Semiconductor

Secure Science Corporation