Kestrel Institute

Kestrel Technology

Using Axe to Reason About Binary Code

Eric Smith

Kestrel Institute

and

Kestrel Technology

ACL2 Workshop, May, 2017

Goal

- Lift binary code into logic
  - JVM bytecode
  - **x86 binary code**

- Then
  - **verify against a spec**
    - **using Axe**
    - or by constructing an APT derivation
  - analyze / prove properties
  - equivalence check two implementations
  - compare to malware
  - run on concrete data

Step 0: Parse the binary

- Parsers for Mach-O and PE (Windows) binaries.
- Build an ACL2 constant representing the binary.

Parsed Mach-O binary for TEA (Tiny Encryption Algorithm)

302 lines total

```
(DEFCONST
 |*wiki*|
 '((:MAGIC . :MH_MAGIC_64)
   (:HEADER (:CPUTYPE . :CPU_TYPE_X86_64)
            (:CPUSUBTYPE . 2147483651)
            (:FILETYPE . :MH_EXECUTE)
            (:NCMDS . 15)
            (:SIZEOFCMDS . 1360)
            (:FLAGS :MH_NOUNDEFS :MH_DYLDLINK
                    :MH_TWOLEVEL :MH_PIE)
            (:RESERVED . 0))
   (:CMDS
    ((:CMD . :LC_SEGMENT_64)
     (:SEGNAME . "__PAGEZERO")
     (:VMADDR . 0)
     (:VMSIZE . 4294967296)
     (:MAXPROT . 0)
     (:INITPROT . 0)
     (:FLAGS)
     (:SECTIONS))
    ((:CMD . :LC_SEGMENT_64)
     (:SEGNAME . "__TEXT")
     (:VMADDR . 4294967296)
     (:VMSIZE . 4096)
     (:MAXPROT . 7)
     (:INITPROT . 5)
     (:FLAGS)
     (:SECTIONS
          ((:SECTNAME . "__text")
           (:TYPE . :S_REGULAR)
           (:SEGNAME . "__TEXT")
           (:ADDR . 4294970560)
           (:ALIGN . 4)
           (:RELOFF . 0)
           (:NRELOC . 0)
           (:ATTRIBUTES :S_ATTR_PURE_INSTRUCTIONS :S_ATTR_SOME_INSTRUCTIONS)
           (:RESERVED1 . 0)
           (:RESERVED2 . 0)
           (:RESERVED3 . 0)
           (:CONTENTS 85 72
                      137 229 72 137 125 248 72 137 117 240 72
                      139 117 248 139 6 137 69 236 72 139 117
                      248 139 70 4 137 69 232 199 69 228 0 0
                      0 0 199 69 220 185 121 55 158 72 139 117
                      240 139 6 137 69 216 72 139 117 240 139
                      70 4 137 69 212 72 139 117 240 139 70 8
                      137 69 208 72 139 117 240 139 70 12 137
                      69 204 199 69 224 0 0 0 0 131 125 224 32
                      15 131 91 0 0 0 139 69 220 3 69 228 137
                      69 228 139 69 232 193 224 4 3 69 216 139
                      77 232 3 77 228 49 200 139 77 232 193
```

Parsed PE (Windows) binary for TEA

32,589 lines total !

```
(DEFCONST
 |*wiki.vs.exe*|
 '((:MS-DOS-STUB 77 90 144 0 3 0
                 0 0 4 0 0 0 255 255 0 0 184 0 0 0 0 0 0
                 0 64 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 240 0
                 0 0 14 31 186 14 0 180 9 205 33 184 1 76
                 205 33 84 104 105 115 32 112 114 111 103
                 114 97 109 32 99 97 110 110 111 116 32
                 98 101 32 114 117 110 32 105 110 32 68
                 79 83 32 109 111 100 101 46 13 13 10 36
                 0 0 0 0 0 0 0 241 80 106 14 181 49 4 93
                 181 49 4 93 181 49 4 93 1 173 245 93 188
                 49 4 93 1 173 247 93 194 49 4 93 1 173
                 246 93 173 49 4 93 104 206 207 93 182
                 49 4 93 181 49 5 93 225 49 4 93 142 111
                 7 92 164 49 4 93 142 111 1 92 168 49 4
                 93 142 111 0 92 164 49 4 93 34 111 0 92
                 180 49 4 93 34 111 6 92 180 49 4 93 82
                 105 99 104 181 49 4 93 0 0 0 0 0 0 0 0)
   (:SIG 80 69 0 0)
   (:COFF-FILE-HEADER
       (:CHARACTERISTICS :IMAGE_FILE_EXECUTABLE_IMAGE :IMAGE_FILE_32BIT_MACHINE)
       (:SIZE-OF-OPTIONAL-HEADER . 224)
       (:NUMBER-OF-SYMBOLS . 0)
       (:POINTER-TO-SYMBOL-TABLE . 0)
       (:TIME-DATE-STAMP . 1494114730)
       (:NUMBER-OF-SECTIONS . 7)
       (:MACHINE . :IMAGE_FILE_MACHINE_I386))
   (:STRING-TABLE . :NONE)
   (:SYMBOL-TABLE . :NONE)
   (:OPTIONAL-HEADER-STANDARD-FIELDS (:BASE-OF-DATA . 323584)
                                     (:BASE-OF-CODE . 4096)
                                     (:ADDRESS-OF-ENTRY-POINT . 5611)
                                     (:SIZE-OF-UNINITIALIZED-DATA . 0)
                                     (:SIZE-OF-INITIALIZED-DATA . 67072)
                                     (:SIZE-OF-CODE . 318464)
                                     (:MINOR-LINKER-VERSION . 0)
                                     (:MAJOR-LINKER-VERSION . 14)
                                     (:MAGIC . :PE32))
   (:OPTIONAL-HEADER-WINDOWS-SPECIFIC-FIELDS
        (:NUMBER-OF-RVA-AND-SIZES . 16)
        (:SIZE-OF-HEAP-COMMIT . 4096)
        (:SIZE-OF-HEAP-RESERVE . 1048576)
        (:SIZE-OF-STACK-COMMIT . 4096)
        (:SIZE-OF-STACK-RESERVE . 1048576)
        (:DLL-CHARACTERISTICS . 33088)
        (:SUBSYSTEM . 3)
        (:CHECK-SUM . 0)
        (:SIZE-OF-HEADERS . 1024)
        (:SIZE-OF-IMAGE . 405504)
```

Axe Tools

- Axe Rewriter
- Axe Prover
- Axe Equivalence Checker
- Lifter: JVM to logic
- Lifter: x86 to logic

- All built on ACL2
- All based on structure-shared terms (DAGs)

Axe Rewriter

- Represents terms as DAGs
  - Represent each sub-term only once
  - Allows massive sharing of structure
  - Can give exponential space/time savings
  - Manipulated using arrays under the hood.
  - Can be embedded in ACL2 terms
- Fast: 600K rewrite rule attempts per sec.
- Fancy features
  - conditional rules
  - assumptions and free variable matching
  - axe-syntaxp, axe-bind-free
  - axe-rewrite-objective
  - “work hard” – like force
  - monitoring rules
  - memoization
  - limited use of content from overarching ifs
  - outside-in rewriting
- No forward chaining, linear, or type-prescription
- Does not produce proofs

```
((28 ACL2::BVPLUS '32 27 14)
 (27 ACL2::SLICE '31 '24 26)
 (26 ACL2::BVMULT '32 '16843009 25)
 (25 ACL2::BVAND '32 24 '252645135)
 (24 ACL2::BVPLUS '32 23 22)
 (23 ACL2::SLICE '31 '4 22)
 (22 ACL2::BVPLUS '32 21 19)
 (21 ACL2::BVAND '30 20 '858993459)
 (20 ACL2::SLICE '31 '2 18)
 (19 ACL2::BVAND '32 18 '858993459)
 (18 ACL2::BVPLUS '32 1 17)
 (17 ACL2::BVUMINUS '32 16)
 (16 ACL2::BVAND '31 15 '1431655765)
 (15 ACL2::SLICE '63 '33 0)
 (14 ACL2::SLICE '31 '24 13)
 (13 ACL2::BVMULT '32 '16843009 12)
 (12 ACL2::BVAND '32 11 '252645135)
 (11 ACL2::BVPLUS '32 10 9)
 (10 ACL2::SLICE '31 '4 9)
 (9 ACL2::BVPLUS '32 8 6)
 (8 ACL2::BVAND '30 7 '858993459)
 (7 ACL2::SLICE '31 '2 5)
 (6 ACL2::BVAND '32 5 '858993459)
 (5 ACL2::BVPLUS '32 0 4)
 (4 ACL2::BVUMINUS '32 3)
 (3 ACL2::BVAND '31 2 '1431655765)
 (2 ACL2::SLICE '31 '1 0)
 (1 ACL2::SLICE '63 '32 0)
 (0 . X))
```

Axe Equivalence Checker

- Tactic-based:
  - Rewriting
  - SMT solving
  - “sweeping and merging”
  - pruning dead branches (with STP and/or rewriting)
  - case-splitting
  - fancy handling of loops/recursions
- Can compare:
  - code to spec
  - code to code

Lifting Into Logic

- JVM Lifter
  - Based on our JVM model
  - Has been used on dozens of examples
  - Can lift loops to recursive functions
- **X86 Lifter**
  - Based on Shilpi's x86 model
  - Newer
  - Support for loops still in progress
- Both lifters use the Axe rewriter for symbolic execution.

Prototype x86 Lifter

- Can lift small x86 binaries into logic
  - subroutine calls
  - conditional branches
  - data from data segment
  - unrollable loops
- Automatically adds lots of standard assumptions
  - especially if there is a symbol table
- Symbolic execution with Axe is orders of magnitude faster than with ACL2's rewriter
- No clock functions!
  - Partial function to "run until return" (run-until-rsp-greater-than)
  - Repeatedly open one step and simplify
- Currently can only lift unrollable loops
  - Loop lifter in progress, based on JVM lifter
- Does not produce proofs
  - Must trust Axe, etc.

Trivial Example: Lifting “add” (Mach-O) into Logic

C function:

```
int add(int x, int y)
  { return(x+y); }
```

Lift the subroutine into logic:

```
(def-lifted-x86 add1 "_add"
        acl2::|*add1.o*| 1)
```

```
add1.o:
(__TEXT,__text) section
_add:
0000000100000ec0        55                  pushq   %rbp
0000000100000ec1        4889e5              movq    %rsp, %rbp
0000000100000ec4        8d0437              leal    (%rdi,%rsi), %eax
0000000100000ec7        5d                  popq    %rbp
0000000100000ec8        c3                  retq
0000000100000ec9        0f1f8000000000      nopl    (%rax)
```

```
(DEFUN-NX
 ADD1 (X86)
 (DECLARE (XARGS :STOBJS X86 :VERIFY-GUARDS NIL))
 (XW
  ':RGF
  '0
  (ACL2::BVPLUS '32
                (XR ':RGF '6 X86)
                (XR ':RGF '7 X86))
  (XW
   ':RGF
   '4
   (BINARY-+ '8 (XR ':RGF '4 X86))
   (XW
    ':RIP
    '0
    (COMBINE-BYTES
     (MV-NTH
          '1
          (RB (CREATE-CANONICAL-ADDRESS-LIST '8
                                             (XR ':RGF '4 X86))
              ':R
              X86)))
    (MV-NTH
     '1
     (WB
      (CREATE-ADDR-BYTES-ALIST
           (CREATE-CANONICAL-ADDRESS-LIST
                '8
                (BINARY-+ '-8 (XR ':RGF '4 X86)))
           (BYTE-IFY '8
                     (ACL2::LOGHEAD$INLINE '64
                                           (XR ':RGF '5 X86))))
      X86))))))
```

Trivial Example: Lifting “add” (PE)

```
(DEFUN
 ADD1 (X86)
 (DECLARE (XARGS :NON-EXECUTABLE T :MODE :LOGIC))
 (DECLARE (XARGS :STOBJS X86 :VERIFY-GUARDS NIL))
 (PROG2$
  (ACL2::THROW-NONEXEC-ERROR 'ADD1
                             (LIST X86))
  (XW
   ':RGF
   '0
   (ACL2::BVPLUS '32
                 (XR ':RGF '2 X86)
                 (XR ':RGF '1 X86))
   (XW
    ':RGF
    '2
    (ACL2::LOGHEAD$INLINE '32
                          (XR ':RGF '1 X86))
    (XW
     ':RGF
     '4
     (BINARY-+ '8 (XR ':RGF '4 X86))
     (XW
      ':RIP
      '0
      (COMBINE-BYTES
       (MV-NTH
         '1
         (RB (CREATE-CANONICAL-ADDRESS-LIST '8
                                            (XR ':RGF '4 X86))
             ':R
             X86)))
      (MV-NTH
       '1
       (WB
        (CREATE-ADDR-BYTES-ALIST
          (CREATE-CANONICAL-ADDRESS-LIST
               '4
               (BINARY-+ '16 (XR ':RGF '4 X86)))
          (BYTE-IFY '4
                    (ACL2::LOGHEAD$INLINE '32
                                          (XR ':RGF '2 X86))))
        (MV-NTH
         '1
         (WB
          (CREATE-ADDR-BYTES-ALIST
               (CREATE-CANONICAL-ADDRESS-LIST
                    '4
                    (BINARY-+ '8 (XR ':RGF '4 X86)))
               (BYTE-IFY
                    '4
                    (ACL2::LOGHEAD$INLINE '32
                                          (XR ':RGF '1 X86))))
          (MV-NTH
           '1
```

```
(WB
 (CREATE-ADDR-BYTES-ALIST
    (CREATE-CANONICAL-ADDRESS-LIST
         '8
         (BINARY-+ '-8 (XR ':RGF '4 X86)))
    (BYTE-IFY
         '8
         (ACL2::LOGHEAD$INLINE '64
                               (XR ':RGF '5 X86))))
 (!FLGI
  '0
  (CF-SPEC32$INLINE
    (BINARY-+
         (ACL2::LOGHEAD$INLINE '32
                               (XR ':RGF '2 X86))
         (ACL2::LOGHEAD$INLINE '32
                               (XR ':RGF '1 X86))))
  (!FLGI
   '2
   (PF-SPEC32$INLINE
        (ACL2::BVPLUS '32
                      (XR ':RGF '2 X86)
                      (XR ':RGF '1 X86)))
   (!FLGI
    '4
    (ADD-AF-SPEC32$INLINE
         (ACL2::LOGHEAD$INLINE '32
                               (XR ':RGF '2 X86))
         (ACL2::LOGHEAD$INLINE '32
                               (XR ':RGF '1 X86)))
    (!FLGI
     '6
     (IF (EQUAL '0
                (ACL2::BVPLUS '32
                              (XR ':RGF '2 X86)
                              (XR ':RGF '1 X86)))
         '1
         '0)
     (!FLGI
      '7
      (ACL2::GETBIT
           '31
           (ACL2::BVPLUS '32
                         (XR ':RGF '2 X86)
                         (XR ':RGF '1 X86)))
      (!FLGI
       '11
       (OF-SPEC32$INLINE
         (BINARY-+ (LOGEXT '32 (XR ':RGF '2 X86))
                   (LOGEXT '32 (XR ':RGF '1 X86))))
       X86)))))))))))))))))
```

Using / Extending the x86 Model

- Adding many rewrite rules
  - Some adjustments for Axe rewriter
  - Rules about disjointness
  - Connecting to our bit vector library
    - Every operator has an explicit size
    - Hundreds of rewrite rules
    - Used in our specs for crypto code
    - Used in translation to STP SMT solver
    - Used in the Axe equivalence checker

- Adding for 32-bit instructions to x86 model.

Examples

- Popcount
- TEA

Example: popcount

- Count the number of 1's in a bit vector
- Optimized C program
  - Correctness non-obvious!
- Lift to a structure-shared "DAG"
- Lifting takes ~1 second.

```c
int popcount_32 (unsigned int v)
{
  v = v - ((v >> 1) & 0x55555555);
  v = (v & 0x33333333) + ((v >> 2) & 0x33333333);
  v = ((v + (v >> 4) & 0xF0F0F0F) * 0x1010101) >> 24;
  return(v);
 }

int popcount_64 (long unsigned int v)
{
  long unsigned int v1, v2;
  // v1: lower 32 bits of v
  v1 = (v & 0xFFFFFFFF);
  // printf ("\n v1: %lu", v1);
  // v2: upper 32 bits of v
  v2 = (v >> 32);
  // printf ("\n v2: %lu", v2);
  return (popcount_32(v1) + popcount_32(v2));
}
```

Example: popcount

```
(def-lifted-x86-axe popcount "_popcount_64" acl2::|*popcount-64.o*|
  1
  :output (:register 0)
  :assumptions '((equal (xr ':rgf '7 x86) x))
  :enable (lifter-rules)
  :print nil)
```

```
int popcount_32 (unsigned int v)
{
  v = v - ((v >> 1) & 0x55555555);
  v = (v & 0x33333333) + ((v >> 2) & 0x33333333);
  v = ((v + (v >> 4) & 0xF0F0F0F) * 0x1010101) >> 24;
  return(v);
 }

int popcount_64 (long unsigned int v)
{
  long unsigned int v1, v2;
  // v1: lower 32 bits of v
  v1 = (v & 0xFFFFFFFF);
  // printf ("\n v1: %lu", v1);
  // v2: upper 32 bits of v
  v2 = (v >> 32);
  // printf ("\n v2: %lu", v2);
  return (popcount_32(v1) + popcount_32(v2));
}
```

Lift

```
((28 ACL2::BVPLUS '32 27 14)
 (27 ACL2::SLICE '31 '24 26)
 (26 ACL2::BVMULT '32 '16843009 25)
 (25 ACL2::BVAND '32 24 '252645135)
 (24 ACL2::BVPLUS '32 23 22)
 (23 ACL2::SLICE '31 '4 22)
 (22 ACL2::BVPLUS '32 21 19)
 (21 ACL2::BVAND '30 20 '858993459)
 (20 ACL2::SLICE '31 '2 18)
 (19 ACL2::BVAND '32 18 '858993459)
 (18 ACL2::BVPLUS '32 1 17)
 (17 ACL2::BVUMINUS '32 16)
 (16 ACL2::BVAND '31 15 '1431655765)
 (15 ACL2::SLICE '63 '33 0)
 (14 ACL2::SLICE '31 '24 13)
 (13 ACL2::BVMULT '32 '16843009 12)
 (12 ACL2::BVAND '32 11 '252645135)
 (11 ACL2::BVPLUS '32 10 9)
 (10 ACL2::SLICE '31 '4 9)
 (9 ACL2::BVPLUS '32 8 6)
 (8 ACL2::BVAND '30 7 '858993459)
 (7 ACL2::SLICE '31 '2 5)
 (6 ACL2::BVAND '32 5 '858993459)
 (5 ACL2::BVPLUS '32 0 4)
 (4 ACL2::BVUMINUS '32 3)
 (3 ACL2::BVAND '31 2 '1431655765)
 (2 ACL2::SLICE '31 '1 0)
 (1 ACL2::SLICE '63 '32 0)
 (0 . X))
```

Example: popcount

- Spec: (acl2::bvcount 64 x)
  - Unrolls to naive algorithm (check each bit and count the 1's)
- Equivalence proof by unrolling spec, rewriting, calling SMT (most work done by SMT).
  - Proof takes a few minutes
- Shows spec and code equivalent, for all 2^{64} inputs.

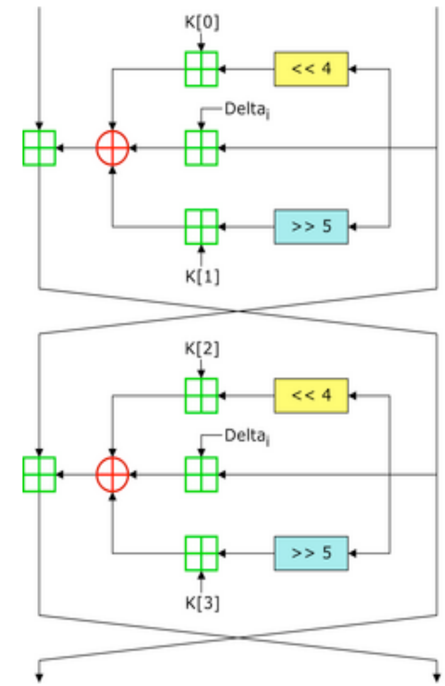
Example: TEA Block Cipher (Tiny Encryption Algorithm)

Formal spec:

```
(defconst *delta* #x9e3779b9)

(defun tea-encrypt-loop (n y z sum k)
  (declare (xargs :guard (and (unsigned-byte-p 32 n) ;n<=32
                              (unsigned-byte-p 32 y)
                              (unsigned-byte-p 32 z)
                              (unsigned-byte-p 32 sum)
                              (bv-arrayp 32 4 k))))
  (if (zp n)
      (mv y z)
    (let* ((n (+ -1 n))
           (sum (bvplus 32 sum *delta*))
           (y (bvplus 32 y (bvxor 32 (bvplus 32 (shl 32 z 4) (bv-array-read 32 4 0 k))
                                   (bvxor 32 (bvplus 32 z sum)
                                          (bvplus 32 (shr 32 z 5) ;unsigned right-shift
                                                  (bv-array-read 32 4 1 k))))))
           (z (bvplus 32 z (bvxor 32 (bvplus 32 (shl 32 y 4) (bv-array-read 32 4 2 k))
                                   (bvxor 32 (bvplus 32 y sum)
                                          (bvplus 32 (shr 32 y 5) ;unsigned right-shift
                                                  (bv-array-read 32 4 3 k)))))))
      (tea-encrypt-loop n y z sum k))))

;; encrypt value V with key K
(defun tea-encrypt (v k)
  (declare (xargs :guard (and (bv-arrayp 32 2 v)
                              (bv-arrayp 32 4 k))))
  (let* ((y (bv-array-read 32 2 0 v))
         (z (bv-array-read 32 2 1 v))
         (sum 0)
         (n 32))
    (mv-let (y z)
      (tea-encrypt-loop n y z sum k)
      (bv-array-write 32 2 0 y (bv-array-write 32 2 1 z '(0 0))))))
```



Example: TEA

```
void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    uint32_t delta=0x9e3779b9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i < 32; i++) {                       /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

- Lifting the binary requires assuming non-overlap in memory of:
  - Params (v, k) and next stack slots
  - Params (v, k) and code
  - v param and stored return address

Example: TEA

- Stats on lifted TEA (after extracting the result):

```
Showing info for DAG:
557 unique nodes
71154924322843234631020052204138214 total nodes
24 Variables:
  IN0 IN1 IN2 IN3 IN4 IN5 IN6 IN7 KEY0 KEY1 KEY2 KEY3
  KEY4 KEY5 KEY6 KEY7 KEY8 KEY9 KEY10 KEY11
  KEY12 KEY13 KEY14 KEY15
5 Functions:
  ACL2::BVCAT ACL2::SLICE ACL2::BVXOR ACL2::BVPLUS
  CONS
Function counts:
CONS:                  2
ACL2::SLICE:          64
ACL2::BVCAT:          83
ACL2::BVXOR:         128
ACL2::BVPLUS:        256
```

- Unrolled spec is similar
- Equivalence proof via rewriting
  - 4,540 rule hits of 229,625 tries
  - 0.23 seconds

```
((556 CONS 546 555)
 (555 CONS 554 'NIL)
 (554 ACL2::BVPLUS '32 553 538)
 (553 ACL2::BVXOR '32 551 552)
 (552 ACL2::BVXOR '32 549 548)
 (551 ACL2::BVPLUS '32 550 41)
 (550 ACL2::SLICE '31 '5 546)
 (549 ACL2::BVPLUS '32 '3337565984 546)
 (548 ACL2::BVPLUS '32 547 38)
 (547 ACL2::BVCAT '28 546 '4 '0)
 (546 ACL2::BVPLUS '32 545 530)
 (545 ACL2::BVXOR '32 543 544)
 (544 ACL2::BVXOR '32 541 540)
 (543 ACL2::BVPLUS '32 542 35)
 (542 ACL2::SLICE '31 '5 538)
 (541 ACL2::BVPLUS '32 '3337565984 538)
 ...
 (29 ACL2::BVCAT '24 28 '8 7)
 (28 ACL2::BVCAT '16 27 '8 6)
 (27 ACL2::BVCAT '8 4 '8 5)
 (26 ACL2::BVCAT '24 25 '8 3)
 (25 ACL2::BVCAT '16 24 '8 2)
 (24 ACL2::BVCAT '8 0 '8 1)
 (23 . KEY15)
 (22 . KEY14)
 (21 . KEY13)
 (20 . KEY12)
 (19 . KEY11)
 (18 . KEY10)
 (17 . KEY9)
 (16 . KEY8)
 (15 . KEY7)
 (14 . KEY6)
 (13 . KEY5)
 (12 . KEY4)
 (11 . KEY3)
 (10 . KEY2)
 (9 . KEY1)
 (8 . KEY0)
 (7 . IN7)
 (6 . IN6)
 (5 . IN5)
 (4 . IN4)
 (3 . IN3)
 (2 . IN2)
 (1 . IN1)
 (0 . IN0))
```

Challenges / Next Steps

- Lifting loops in x86 binaries
  - Approach similar to our JVM lifter
  - May do some things differently:
    - Have lifted functions still traffic in x86 memories
      - Don’t require all aliasing to be resolved
    - Allow lifted functions to represent exceptions / errors
      - Don’t require proving absence of errors

Bonus Example: TEA in Java

TEA in Java (bouncycastle)

```
private int encryptBlock(
    byte[]  in,
    int     inOff,
    byte[]  out,
    int     outOff)
{
    // Pack bytes into integers
    int v0 = bytesToInt(in, inOff);
    int v1 = bytesToInt(in, inOff + 4);

    int sum = 0;

    for (int i = 0; i != rounds; i++)
    {
        sum += delta;
        v0  += ((v1 << 4) + _a) ^ (v1 + sum) ^ ((v1 >>> 5) + _b);
        v1  += ((v0 << 4) + _c) ^ (v0 + sum) ^ ((v0 >>> 5) + _d);
    }

    unpackInt(v0, out, outOff);
    unpackInt(v1, out, outOff + 4);

    return block_size;
}
```

TEA in Java

- Lifting into logic
- Reconstruct a derivation
  - Proof-emitting transformation steps
  - Link the code and the spec