WHAT DO WE WANT?

Rich Salz

WHO IS WE?

- IETF – standards org
- Browsers – dominant platform
- Open source -- developers

But First

- Who are you?
  - New algorithm
  - New security properties*
  - New implementation
  - Want to get it used
- Did I miss anything?

*Constant time…

IETF

- Main areas
  - TLS
  - Research; post-quantum
  - Vehicle to vehicle
  - IoT
- Other stuff – DNS privacy, BGP routing?

IETF Structure

- Divided into areas, each with 1-2 Area Directors:
  - Security, Applications, DNS, Routing
- Working groups … do the work
  - Write, review, discuss documents; grow to be RFC's
- AD's appoint WG chairs, WG chairs appoint doc editors

IETF Workflow

- Someone(s) writes a draft
- WG adopts it; it now “belongs” to the IETF
- WG works on it (email and 3x/year F2F)
- WG does “last call”
- IESG/IETF does “last call”
  - Any AD can raise a DISCUSS item, back to drawing board
- Voilà it’s a standard

IRTF?

- IETF focus is on getting something that works deployed
- IRTF is focused on research to feed into #1
- Major areas include
  - CFRG – crypto forum
  - T2TRG – thing to thing

- They give out a prize (USD$500, trip to IETF, etc)

CFRG

- Becoming the “think tank” of TLS and other security WG’s
- Co-chairs Kenny Paterson and Alexey Melnikov

- Just started a “reading group” to review papers that people (IETF community) think are of interest

IETF, CFRG Mindset

- National-scale attackers exist and are bad
- National standards are not good *a priori*
- ISO standards are not good *a priori*
- Proofs are important
- "Hard" should be strictly defined
- Post-quantum becoming important
  - Proceeding cautiously

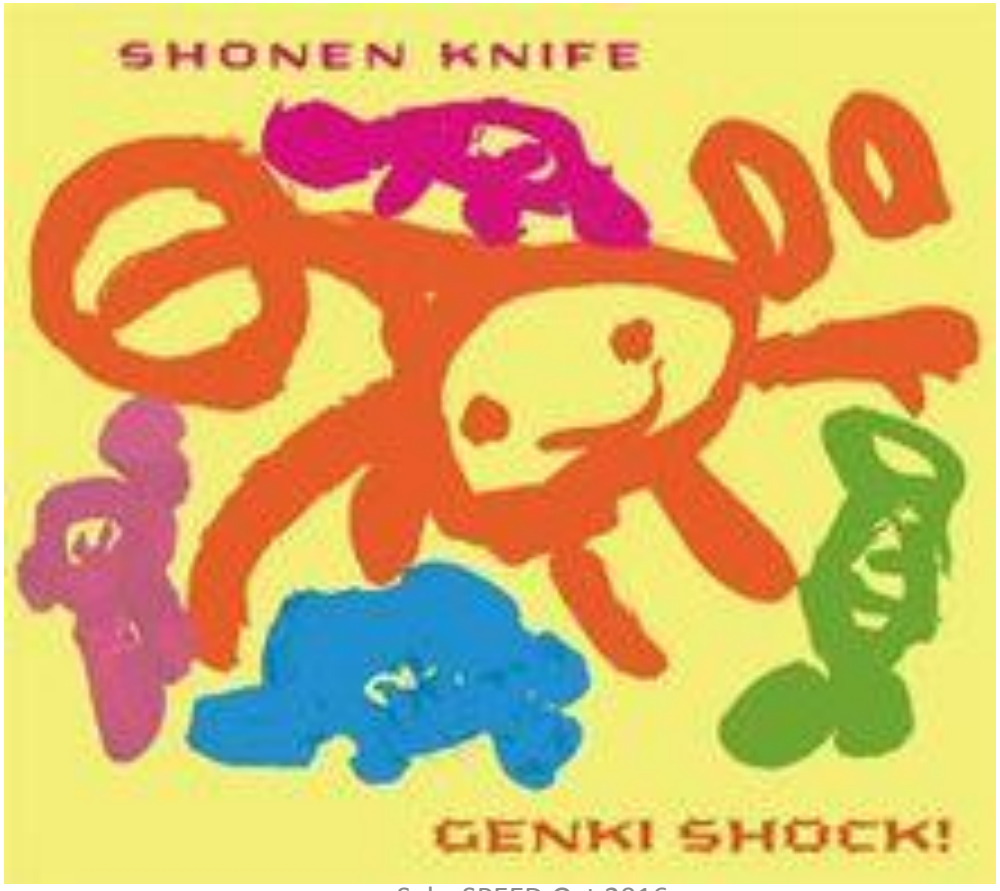
Browsers

- The usual suspects
  - Chrome
  - Firefox
  - Microsoft (IE, Edge)
  - Apple

- They cooperate; e.g., SHA-1 deprecation

Browsers

- “Don’t quote me”



OpenSource

- Smash it into a kernel
- OpenSSH
- Throw it up on GitHub*
- OpenSSL**

Throw it up on GitHub

- Simple
  - All the infrastructure is there
- Someone *may* use it
- If it's tuning, "fork" the base application

OpenSSL

- The major open source crypto toolkit
- Was crap, is better
- All development on GitHub now
- Make pull requests
- Soon to be Apache license

OpenSSL Data Types

- Portable:
  - Written in C
  - Runs everywhere
  - Assembler versions of some parts (24+ platforms)
- Common (opaque) structures for
  - ASN.1 types
  - Asymmetric keys
  - Enveloping operations (encrypt, hmac, decrypt; AEAD missing)

OpenSSL Assembler

- C compilers are too smart/clever/evil
- Zero out memory
- Constant-time (AES)
- New hardware instructions not always suported
  - Run-time detection possible/desired

CPU types supported

- x86, x86_64 (ASM, MASM, etc.)
- ARM (v6, v7, v8, etc.)
- PPC-32, -64

- MIPS-32, -64
- System/390
- PA-RISC
- SPARC

PerlASM

```
$code.=<<___;
.text
.extern OPENSSL_ia32cap_P

.globl  aesni_cbc_sha1_enc
.type   aesni_cbc_sha1_enc,\@abi-omnipotent
.align  32
aesni_cbc_sha1_enc:
      # caller should check for SSSE3 and AES-NI bits
      mov     OPENSSL_ia32cap_P+0(%rip),%r10d
      mov     OPENSSL_ia32cap_P+4(%rip),%r11

___
$code.=<<___ if ($shaext);
      bt      \$61,%r11             # check SHA bit
      jc      aesni_cbc_sha1_enc_shaext

___
$code.=<<___ if ($avx);
      and     \$`1<<28`,%r11d       # mask AVX bit
      and     \$`1<<30`,%r10d       # mask "Intel CPU" bit
      or      %r11d,%r10d
      cmp     \$`1<<28|1<<30`,%r10d
      je      aesni_cbc_sha1_enc_avx

___
```

PerlASM, cont'd

```
sub AUTOLOAD()          # thunk [simplified] 32-bit style perlasm
{ my $opcode = $AUTOLOAD; $opcode =~ s/.*::///;
  my $arg = pop;
    $arg = "\$$arg" if ($arg*1 eq $arg);
    $code .= "\t$opcode\t".join(',',$arg,reverse @_)."\n";
}

@x=("%eax","%ebx","%ecx","%edx",map("%r${_}d",(8..11)),
    "%nox","%nox","%nox","%nox",map("%r${_}d",(12..15)));
@t=("%esi","%edi");

sub ROUND {                 # critical path is 24 cycles per round
my ($a0,$b0,$c0,$d0)=@_;
my ($a1,$b1,$c1,$d1)=map(($_&~3)+(($_+1)&3),($a0,$b0,$c0,$d0));
my ($a2,$b2,$c2,$d2)=map(($_&~3)+(($_+1)&3),($a1,$b1,$c1,$d1));
my ($a3,$b3,$c3,$d3)=map(($_&~3)+(($_+1)&3),($a2,$b2,$c2,$d2));
my ($xc,$xc_)=map("\"$_\"",@t);
my @x=map("\"$_\"",@x);
```

PerlASM, OMG

```
# Normally instructions would be interleaved to favour in-order
 # execution. Generally out-of-order cores manage it gracefully,
 # but not this time for some reason. As in-order execution
 # cores are dying breed, old Atom is the only one around,
 # instructions are left uninterleaved. Besides, Atom is better
 # off executing 1xSSSE3 code anyway...

 (
 "&add   (@x[$a0],@x[$b0])",    # Q1
 "&xor   (@x[$d0],@x[$a0])",
 "&rol   (@x[$d0],16)",
  "&add  (@x[$a1],@x[$b1])",    # Q2
  "&xor  (@x[$d1],@x[$a1])",
  "&rol  (@x[$d1],16)",
```

Thanks!