High-assurance crypto software

Daniel J. Bernstein, Tanja Lange

University of Illinois at Chicago,
Ruhr University Bochum;
Eindhoven University of Technology

Minerva attack can recover private keys from smart cards, cryptographic libraries

Older Athena IDProtect smart cards are impacted, along with the WolfSSL, MatrixSSL, Crypto++, Oracle SunEC, and Libgcrypt crypto libraries.

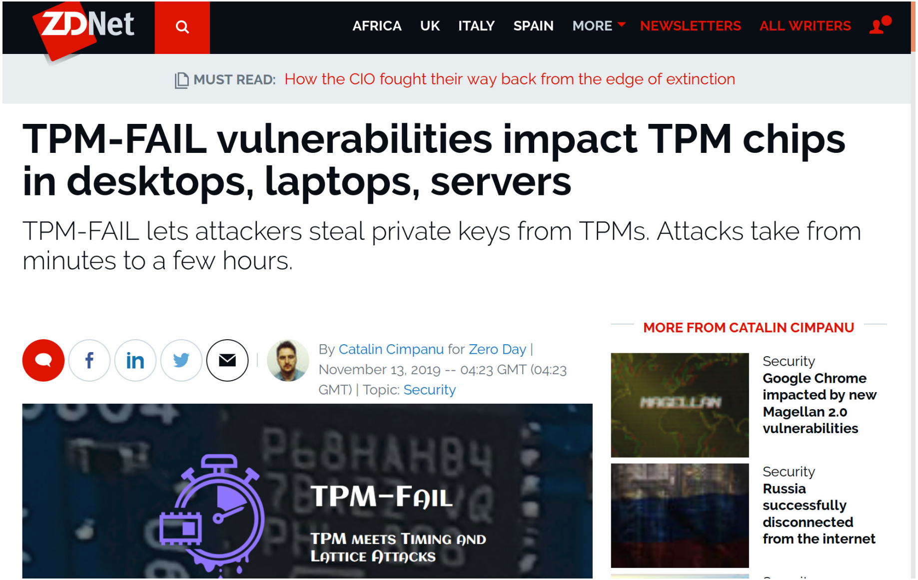
By Catalin Cimpanu for Zero Day | October 3, 2019 -- 12:54 GMT (13:54 BST) | Topic: Security

MORE FROM CATALIN CIMPANU

Security
Google Chrome impacted by new Magellan 2.0 vulnerabilities

Security
Russia successfully disconnected from the internet

ZDNet article



MUST READ: How the CIO fought their way back from the edge of extinction

TPM-FAIL vulnerabilities impact TPM chips in desktops, laptops, servers

TPM-FAIL lets attackers steal private keys from TPMs. Attacks take from minutes to a few hours.

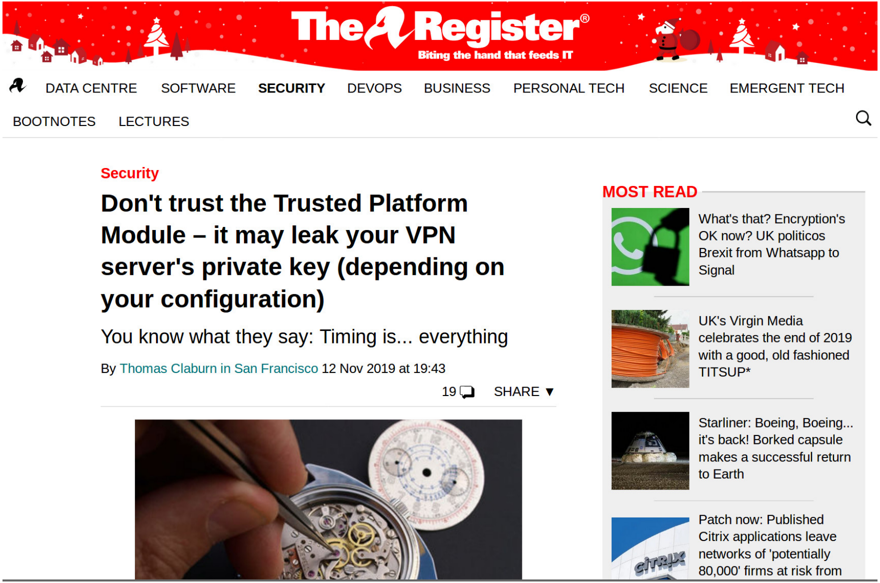
By Catalin Cimpanu for Zero Day | November 13, 2019 -- 04:23 GMT (04:23 GMT) | Topic: Security

MORE FROM CATALIN CIMPANU

Security
Google Chrome impacted by new Magellan 2.0 vulnerabilities

Security
Russia successfully disconnected from the internet

ZDNet article



Security

Don't trust the Trusted Platform Module – it may leak your VPN server's private key (depending on your configuration)

You know what they say: Timing is... everything

By Thomas Claburn in San Francisco 12 Nov 2019 at 19:43

MOST READ

What's that? Encryption's OK now? UK politicos Brexit from Whatsapp to Signal

UK's Virgin Media celebrates the end of 2019 with a good, old fashioned TITSUP*

Starliner: Boeing, Boeing... it's back! Borked capsule makes a successful return to Earth

Patch now: Published Citrix applications leave networks of 'potentially 80,000' firms at risk from

Register article

ELLIPTISCHE KURVEN

Minerva-Angriff zielt auf zertifizierte Krypto-Chips

Forscher konnten zeigen, wie sie mit einem Timing-Angriff die privaten Schlüssel von Signaturen mit elliptischen Kurven auslesen konnten. Verwundbar sind Chips, deren Sicherheit eigentlich zertifiziert wurde.

4. Oktober 2019, 13:41 Uhr, Hanno Böck

Golem article

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.

Try `CAA`, `CBB`, `CCC`, . . .

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `COO` takes slightly longer to fail.
Try `COA`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `COO` takes slightly longer to fail.
Try `COA`, `COB`,

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `COO` takes slightly longer to fail.
Try `COA`, `COB`, `COC`, . . .

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `COO` takes slightly longer to fail.
Try `COA`, `COB`, `COC`, . . . , `CON` takes slightly longer to fail.
⋮

Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:

Try `AAA`, `BBB`, `CCC` takes slightly longer to fail.
Try `CAA`, `CBB`, `CCC`, . . . , `COO` takes slightly longer to fail.
Try `COA`, `COB`, `COC`, . . . , `CON` takes slightly longer to fail.
⋮
Password is `CONGRESS`.

1974: Exploit developed by Alan Bell for TENEX operating system.

Exponentiation with secret exponent (RSA, DH)

Compute c^d given c and d.

```
n = 1000001
d = 12473
c = 41241
l = d.nbits()
D = d.bits()
m = c
for i in range(l-2,-1,-1):
  m = m^2 % n
  if D[i] == 1:
    m = m * c % n
print(m)
```

Exponentiation with secret exponent (RSA, DH)

Compute c^d given c and d.

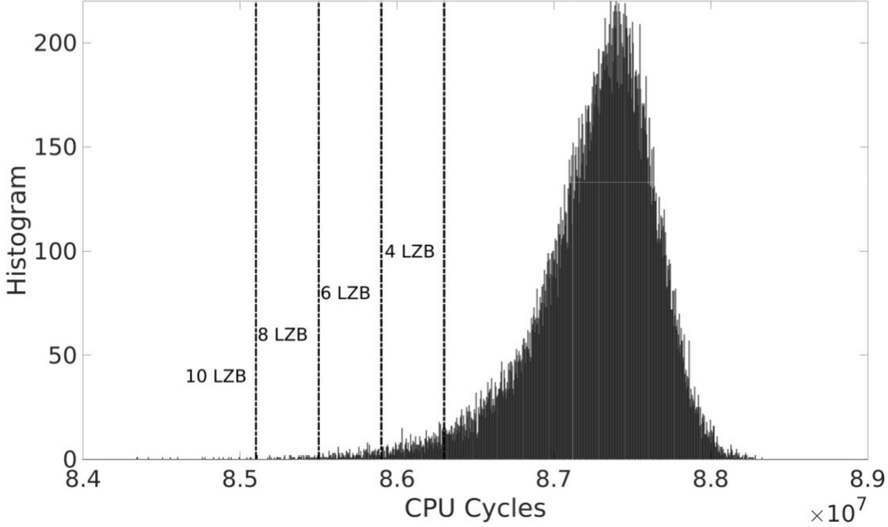
```
n = 1000001
d = 12473
c = 41241
l = d.nbits()
D = d.bits()
m = c
for i in range(l-2,-1,-1): # loop length depends on d
  m = m^2 % n
  if D[i] == 1:
    m = m * c % n
print(m)
```

Exponentiation with secret exponent (RSA, DH)

Compute c^d given c and d.

```
n = 1000001
d = 12473
c = 41241
l = d.nbits()
D = d.bits()
m = c
for i in range(l-2,-1,-1): # loop length depends on d
  m = m^2 % n
  if D[i] == 1: # branch depends on d
    m = m * c % n
print(m)
```

Timings of scalar multiplication on NIST P-256



(Picture from TPM-Fail)

Other exponentiation methods

- The timing variation depends strongly on the length of the scalar/exponent.
- Very sparse or very dense scalars will be miscategorized.
- Faster methods reduce the number of multiplications by using windows: $14019 =$

Other exponentiation methods

- The timing variation depends strongly on the length of the scalar/exponent.
- Very sparse or very dense scalars will be miscategorized.
- Faster methods reduce the number of multiplications by using windows: $14019 =$ `0x36C3` $=$ `0011 0110 1100 0011`

Other exponentiation methods

- The timing variation depends strongly on the length of the scalar/exponent.
- Very sparse or very dense scalars will be miscategorized.
- Faster methods reduce the number of multiplications by using windows: $14019 = \texttt{0x36C3} = \underbrace{\texttt{0011}}_{0\ \ 3}\ \underbrace{\texttt{0110}}_{1\ \ 2}\ \underbrace{\texttt{1100}}_{3\ \ 0}\ \underbrace{\texttt{0011}}_{0\ \ 3}$

Other exponentiation methods

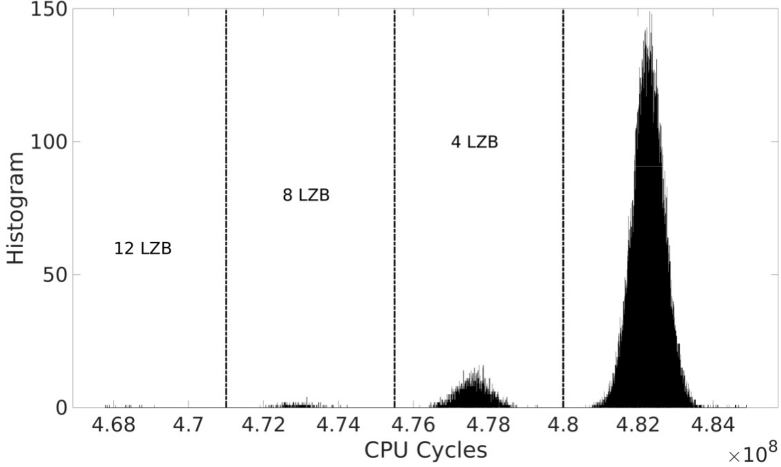
- The timing variation depends strongly on the length of the scalar/exponent.
- Very sparse or very dense scalars will be miscategorized.
- Faster methods reduce the number of multiplications by using windows: $14019 = \texttt{0x36C3} = \underbrace{\texttt{0011}}_{0\ \ 3}\ \underbrace{\texttt{0110}}_{1\ \ 2}\ \underbrace{\texttt{1100}}_{3\ \ 0}\ \underbrace{\texttt{0011}}_{0\ \ 3}$

  Precompute c, c^2, and c^3.

$$c^{14019} = \left(\left(\left(\left(\left(\left(c^3\right)^4 \cdot c\right)^4 \cdot c^2\right)^4 \cdot c^3\right)^4\right)^4\right)^4 \cdot c^3.$$

Same number of squarings, 4 instead of 7 multiplications.

Timings of scalar multiplication on NIST P-256



Larger windows reduce the variability through branching but accentuate the length.

(Picture from TPM-Fail)

How much can a few bits do?

- A bit for RSA, DH, etc.

How much can a few bits do?

- A bit for RSA, DH, etc. More for RSA with CRT decryption.

How much can a few bits do?

- A bit for RSA, DH, etc. More for RSA with CRT decryption.
- A lot for DSA and ECDSA signatures:
  - TPM–Fail: TPM meets Timing and Lattice Attacks
    Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, Nadia Heninger
    `https://tpm.fail/`
  - Minerva attack
    Jan Jancar, Petr Svenda, Vladimir Sedlacek
    `https://minerva.crocs.fi.muni.cz/`

  With just a small bias in the nonces (one-time scalars) the secret signing key leaks.

How much can a few bits do?

- A bit for RSA, DH, etc. More for RSA with CRT decryption.
- A lot for DSA and ECDSA signatures:
  - TPM–Fail: TPM meets Timing and Lattice Attacks
    Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, Nadia Heninger
    `https://tpm.fail/`
  - Minerva attack
    Jan Jancar, Petr Svenda, Vladimir Sedlacek
    `https://minerva.crocs.fi.muni.cz/`

  With just a small bias in the nonces (one-time scalars) the secret signing key leaks.
- Lots of libraries, smart cards, and TPMs affected.

How much can a few bits do?

- A bit for RSA, DH, etc. More for RSA with CRT decryption.
- A lot for DSA and ECDSA signatures:
  - TPM–Fail: TPM meets Timing and Lattice Attacks
    Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, Nadia Heninger
    `https://tpm.fail/`
  - Minerva attack
    Jan Jancar, Petr Svenda, Vladimir Sedlacek
    `https://minerva.crocs.fi.muni.cz/`

  With just a small bias in the nonces (one-time scalars) the secret signing key leaks.
- Lots of libraries, smart cards, and TPMs affected.
- Even worse: hyperthreading attacks, cache-timing attacks, etc. give more fine-grained timing information ⇒ more exploits.

Constant-time exponentiation

```
n = 1000001
d = 12473
c = 41241
l = n.nbits()
D = d.digits(2,padto = l)
m = 1 # so initial squarings don't matter
for i in range(l-1,-1,-1): # fixed-length loop
  m = m^2 % n
  h = m * c % n
  m = (1 - D[i]) * m + D[i] * h # selection by arithmetic
print(m)
```

This costs 1 multiplication per bit, so as slow as worst case.

Interplay with elliptic-curve formulas

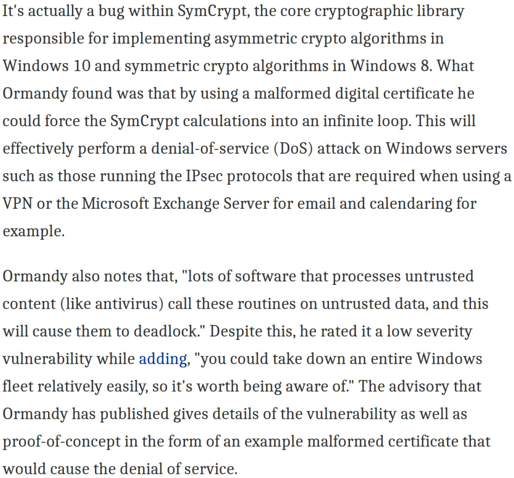
- We can translate this to scalar multiplication on elliptic curves: Initialize with the neutral element, for every bit compute a doubling and an addition.
- Formulas for addition on Weierstrass curves have exceptions for adding ∞, so initialization at ∞ does not work.
- Edwards curves have a complete addition law, **easy** to double or add the neutral element $(0, 1)$.
- The Montgomery ladder has a similar data flow, but the costs per bit of the scalar are **less** than one addition plus one doubling for Montgomery curves.

For more see `https://ecchacks.cr.yp.to`.

210,878 views | Jun 12, 2019, 08:10am

Warning: Google Researcher Drops Windows 10 Zero-Day Security Bomb

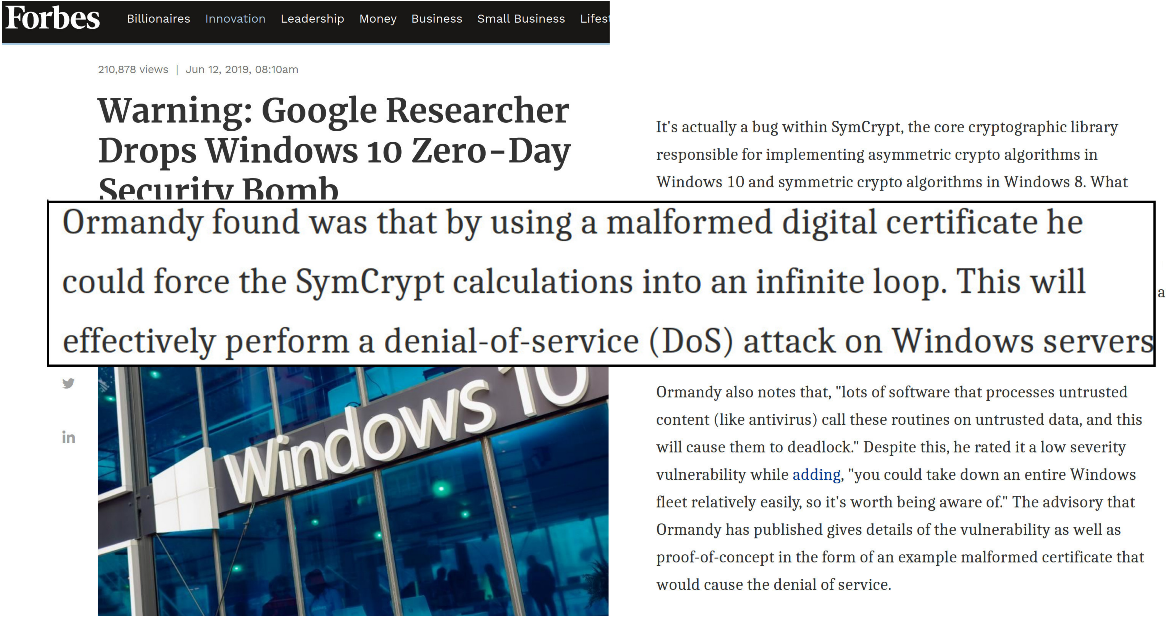
Davey Winder Senior Contributor
Cybersecurity
I report and analyse breaking cybersecurity and privacy stories



It's actually a bug within SymCrypt, the core cryptographic library responsible for implementing asymmetric crypto algorithms in Windows 10 and symmetric crypto algorithms in Windows 8. What Ormandy found was that by using a malformed digital certificate he could force the SymCrypt calculations into an infinite loop. This will effectively perform a denial-of-service (DoS) attack on Windows servers such as those running the IPsec protocols that are required when using a VPN or the Microsoft Exchange Server for email and calendaring for example.

Ormandy also notes that, "lots of software that processes untrusted content (like antivirus) call these routines on untrusted data, and this will cause them to deadlock." Despite this, he rated it a low severity vulnerability while adding, "you could take down an entire Windows fleet relatively easily, so it's worth being aware of." The advisory that Ormandy has published gives details of the vulnerability as well as proof-of-concept in the form of an example malformed certificate that would cause the denial of service.

Forbes article



210,878 views | Jun 12, 2019, 08:10am

Warning: Google Researcher Drops Windows 10 Zero-Day Security Bomb

It's actually a bug within SymCrypt, the core cryptographic library responsible for implementing asymmetric crypto algorithms in Windows 10 and symmetric crypto algorithms in Windows 8. What

Ormandy found was that by using a malformed digital certificate he could force the SymCrypt calculations into an infinite loop. This will effectively perform a denial-of-service (DoS) attack on Windows servers

a

Ormandy also notes that, "lots of software that processes untrusted content (like antivirus) call these routines on untrusted data, and this will cause them to deadlock." Despite this, he rated it a low severity vulnerability while adding, "you could take down an entire Windows fleet relatively easily, so it's worth being aware of." The advisory that Ormandy has published gives details of the vulnerability as well as proof-of-concept in the form of an example malformed certificate that would cause the denial of service.

Forbes article

Using Valgrind to check for secret branches/addresses

```
#include <stdlib.h>
#include <openssl/rc4.h>

int main()
{
  RC4_KEY expandedkey;
  unsigned char *key = malloc(32);
  if (!key) abort();
  RC4_set_key(&expandedkey,32,key);
  free(key);
  return 0;
}
```

Using Valgrind to check for secret branches/addresses

```
$ valgrind ./rc4
==2599== Memcheck, a memory error detector
==2599== Copyright (C) 2002-2017, and GNU GPL'd, by Julia
==2599== Using Valgrind-3.14.0 and LibVEX; rerun with -h f
==2599== Command: ./rc4
==2599==
==2599== Use of uninitialised value of size 8
==2599==    at 0x4A1A0EF: RC4_set_key (in /usr/lib/x86_64-
==2599==    by 0x1090BD: main (in /home/.../rc4)
...
==2599== ERROR SUMMARY: 256 errors from 1 contexts (suppre
```

All good now?

Now we have constant-time exponentation / scalar multiplication if

All good now?

Now we have constant-time exponentation / scalar multiplication if

- the arithmetic is implemented in constant time

All good now?

Now we have constant-time exponentation / scalar multiplication if

- the arithmetic is implemented in constant time
- the processor provides constant-time arithmetic instructions.

All good now?

Now we have constant-time exponentation / scalar multiplication if

- the arithmetic is implemented in constant time
- the processor provides constant-time arithmetic instructions.

Single-clock-cycle instructions are probably OK.

ARM Cortex-M3

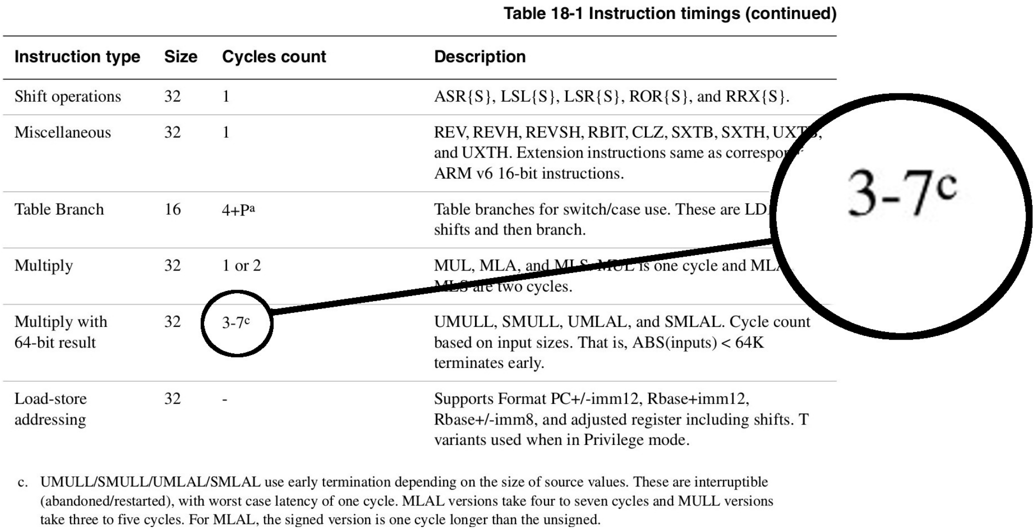


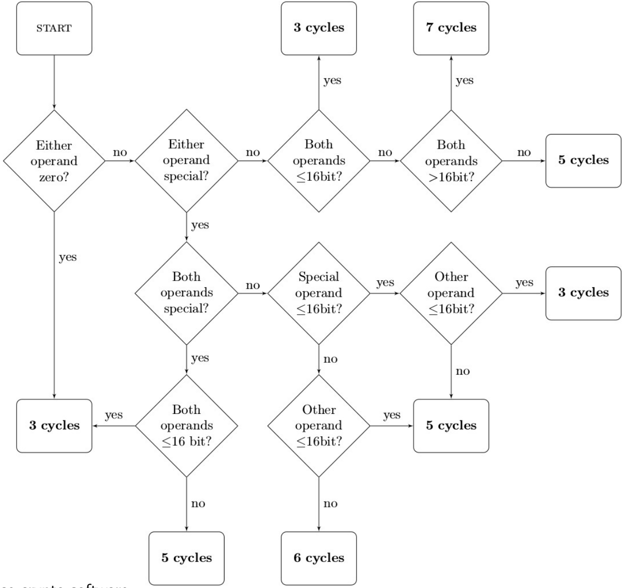
Table 18-1 Instruction timings (continued)

Instruction type	Size	Cycles count	Description
Shift operations	32	1	ASR{S}, LSL{S}, LSR{S}, ROR{S}, and RRX{S}.
Miscellaneous	32	1	REV, REVH, REVSH, RBIT, CLZ, SXTB, SXTH, UXTB, and UXTH. Extension instructions same as correspo ARM v6 16-bit instructions.
Table Branch	16	4+P[a]	Table branches for switch/case use. These are LD shifts and then branch.
Multiply	32	1 or 2	MUL, MLA, and MLS. MUL is one cycle and MLA MLS are two cycles.
Multiply with 64-bit result	32	3-7[c]	UMULL, SMULL, UMLAL, and SMLAL. Cycle count based on input sizes. That is, ABS(inputs) < 64K terminates early.
Load-store addressing	32	-	Supports Format PC+/-imm12, Rbase+imm12, Rbase+/-imm8, and adjusted register including shifts. T variants used when in Privilege mode.

c. UMULL/SMULL/UMLAL/SMLAL use early termination depending on the size of source values. These are interruptible (abandoned/restarted), with worst case latency of one cycle. MLAL versions take four to seven cycles and MULL versions take three to five cycles. For MLAL, the signed version is one cycle longer than the unsigned.

Cortex-M3 Technical Reference Manual - ARM architecture

ARM Cortex-M3 – what does it really do?



Flow chart for UMLAL (unsigned multiply add) from A performance study of X25519 on Cortex-M3 and M4 by Wouter de Groot.

CVE-2018-0733, an OpenSSL bug

"Because of an implementation bug the PA-RISC `CRYPTO_memcmp` function is effectively reduced to only comparing the least significant bit of each byte." Bug introduced May 2016.

CVE-2018-0733, an OpenSSL bug

"Because of an implementation bug the PA-RISC `CRYPTO_memcmp` function is effectively reduced to only comparing the least significant bit of each byte." Bug introduced May 2016.

How severe is this? "This allows an attacker to forge messages that would be considered as authenticated in an amount of tries lower than that guaranteed by the security claims of the scheme."

CVE-2018-0733, an OpenSSL bug

“Because of an implementation bug the PA-RISC `CRYPTO_memcmp` function is effectively reduced to only comparing the least significant bit of each byte.” Bug introduced May 2016.

How severe is this? “This allows an attacker to forge messages that would be considered as authenticated in an amount of tries lower than that guaranteed by the security claims of the scheme.”

— Yes, 2^{16} is “lower than” 2^{128}.

CVE-2017-3738, another OpenSSL bug

Don't care about PA-RISC? How about Intel?

"There is an overflow bug in the AVX2 Montgomery multiplication procedure used in exponentiation with 1024-bit moduli."
Bug introduced July 2013.

CVE-2017-3738, another OpenSSL bug

Don’t care about PA-RISC? How about Intel?

“There is an overflow bug in the AVX2 Montgomery multiplication procedure used in exponentiation with 1024-bit moduli.”
Bug introduced July 2013.

“Attacks against DH1024 are considered just feasible”

CVE-2017-3738, another OpenSSL bug

Don’t care about PA-RISC? How about Intel?

“There is an overflow bug in the AVX2 Montgomery multiplication procedure used in exponentiation with 1024-bit moduli.”
Bug introduced July 2013.

“Attacks against DH1024 are considered just feasible”
— How much time? How much hardware?

CVE-2017-3738, continued

Are you safe if you aren't using DH1024? "Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

CVE-2017-3738, continued

Are you safe if you aren’t using DH1024? “Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely.”

— Really? How much public scrutiny
has the actual computation received from cryptanalysts?

CVE-2017-3738, continued

Are you safe if you aren't using DH1024? "Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

— Really? How much public scrutiny
has the actual computation received from cryptanalysts?

Imagine someone saying "We have analyzed our new cryptosystem and concluded that attacks are not likely."

CVE-2017-3738, continued

Are you safe if you aren't using DH1024? "Analysis suggests that attacks against RSA and DSA as a result of this defect would be very difficult to perform and are not believed likely."

— Really? How much public scrutiny
has the actual computation received from cryptanalysts?

Imagine someone saying "We have analyzed our new cryptosystem and concluded that attacks are not likely."

6 December 2019: Similar OpenSSL advisory for CVE-2019-1551.

Part of the CVE-2017-3738 patch

```
@@ -1093,7 +1093,9 @@
  vmovdqu  -8+32*2-128($ap),$TEMP2

  mov      $r1, %rax
+ vpblendd \$0xfc, $ZERO, $ACC9, $ACC9 # correct $ACC3
  imull    $n0, %eax
+ vpaddq   $ACC9,$ACC4,$ACC4           # correct $ACC3
  and      \$0x1ffffff, %eax

  imulq    16-128($ap),%rbx
@@ -1329,15 +1331,12 @@
```

September 2019: bug announced in Falcon software

Falcon: signature system in round 2 of post-quantum competition.

"The consequences of these bugs are the following:

- Produced signatures were valid but **leaked information on the private key**. [emphasis added]
- Performance was artificially inflated: . . .

The fact that these bugs existed in the first place shows that the traditional development methodology (i.e. 'being super careful') has failed."

Cryptography is notoriously hard to review

Mathematical complications in cryptography lead to subtle bugs.

Cryptography is notoriously hard to review

Mathematical complications in cryptography lead to subtle bugs.

Side-channel countermeasures add more complexity.

Cryptography is notoriously hard to review

Mathematical complications in cryptography lead to subtle bugs.

Side-channel countermeasures add more complexity.

Post-quantum cryptography: even more complex.

Cryptography is notoriously hard to review

Mathematical complications in cryptography lead to subtle bugs.

Side-channel countermeasures add more complexity.

Post-quantum cryptography: even more complex.

Cryptography is applied to large volumes of data.
Often individual cryptographic computations are time-consuming.
Pursuit of speed ⇒ many different cryptographic systems, and
cryptographic code optimized in many ways for particular CPUs.

Cryptography is notoriously hard to review

Mathematical complications in cryptography lead to subtle bugs.

Side-channel countermeasures add more complexity.

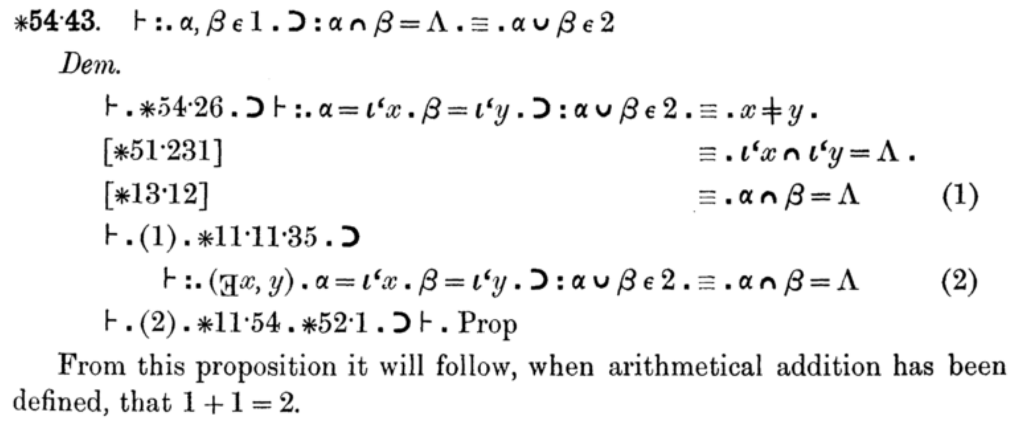
Post-quantum cryptography: even more complex.

Cryptography is applied to large volumes of data.
Often individual cryptographic computations are time-consuming.
Pursuit of speed ⇒ many different cryptographic systems, and cryptographic code optimized in many ways for particular CPUs.

e.g. Keccak Code Package: >20 implementations of SHA-3.
e.g. Google added hand-written Cortex-A7 asm to Linux kernel for Speck128/128-XTS, then switched to (faster) Adiantum-XChaCha.

Formal logic to the rescue?

Whitehead and Russell, *Principia Mathematica*, volume 1, 1st edition (1910), page 379:



$*54\cdot43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54\cdot26 . \supset \vdash :. \alpha = \iota\text{‘}x . \beta = \iota\text{‘}y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[*51\cdot231] \qquad \equiv . \iota\text{‘}x \cap \iota\text{‘}y = \Lambda .$

$[*13\cdot12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11\cdot11\cdot35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota\text{‘}x . \beta = \iota\text{‘}y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . *11\cdot54 . *52\cdot1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these proof-checking tools today.)

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these proof-checking tools today.)

Proving crypto code correct is tedious but not impossible.
Latest EverCrypt release: verified software for Curve25519, Ed25519, ChaCha20, Poly1305, AES-CTR (if CPU has AES-NI), AES-GCM (same), MD5, SHA-1, SHA-2, SHA-3, BLAKE2.

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these proof-checking tools today.)

Proving crypto code correct is tedious but not impossible.
Latest EverCrypt release: verified software for Curve25519, Ed25519, ChaCha20, Poly1305, AES-CTR (if CPU has AES-NI), AES-GCM (same), MD5, SHA-1, SHA-2, SHA-3, BLAKE2.

Good: High confidence that subtle bugs are gone (in the code; but worry about bugs in compiler, CPU, . . .).

Formal verification today

Require code reviewer to *prove* correctness.
Require proofs to pass a proof-checking tool.
(Mathematicians rarely use these proof-checking tools today.)

Proving crypto code correct is tedious but not impossible.
Latest EverCrypt release: verified software for Curve25519, Ed25519, ChaCha20, Poly1305, AES-CTR (if CPU has AES-NI), AES-GCM (same), MD5, SHA-1, SHA-2, SHA-3, BLAKE2.

Good: High confidence that subtle bugs are gone (in the code; but worry about bugs in compiler, CPU, . . .).
Bad: Tons of effort for each implementation.
e.g. EverCrypt doesn't have fast software for smartphone CPUs.

Testing

Testing is great. Test everything. Design for tests.

Why wasn't the PA-RISC `CRYPTO_memcmp` software in OpenSSL
run through millions of tests on random inputs?
And tests on inputs differing in just a few positions?
SUPERCOP crypto test framework has always done this.

Testing

Testing is great. Test everything. Design for tests.

Why wasn't the PA-RISC `CRYPTO_memcmp` software in OpenSSL
run through millions of tests on random inputs?
And tests on inputs differing in just a few positions?
SUPERCOP crypto test framework has always done this.

Good reaction to a bug:
"How can I build fast automated tests to catch this kind of bug?"
Even better to ask question before bug happens.

The most important complaint about testing

Testing can miss attacker-triggerable bugs for rare inputs.

The most important complaint about testing

Testing can miss attacker-triggerable bugs for rare inputs.

e.g. November 2019 paper from Nath and Sarkar points out bugs with probability $\approx 1/2^{64}$ in the fastest code for Curve448:

“On certain kinds of inputs, the code will lead to overflow conditions and hence to incorrect results.

The most important complaint about testing

Testing can miss attacker-triggerable bugs for rare inputs.

e.g. November 2019 paper from Nath and Sarkar points out bugs with probability $\approx 1/2^{64}$ in the fastest code for Curve448:

"On certain kinds of inputs, the code will lead to overflow conditions and hence to incorrect results. This, however, is a very low probability event and cannot be captured using some randomly generated known answer tests (KATs). . . .

The most important complaint about testing

Testing can miss attacker-triggerable bugs for rare inputs.

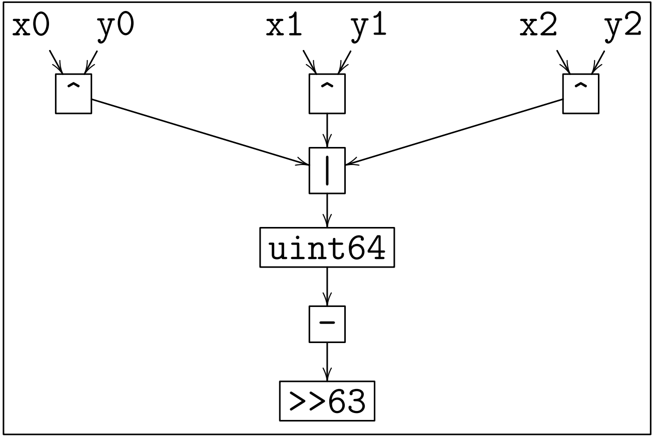
e.g. November 2019 paper from Nath and Sarkar points out bugs with probability $\approx 1/2^{64}$ in the fastest code for Curve448:

"On certain kinds of inputs, the code will lead to overflow conditions and hence to incorrect results. This, however, is a very low probability event and cannot be captured using some randomly generated known answer tests (KATs). . . . We believe that it is important to have proofs of correctness of the reduction algorithms to ensure that the algorithms works correctly for all possible inputs."

Symbolic testing: beyond testing particular inputs

```
.globl CRYPTO_memcmp
CRYPTO_memcmp:
xor    %rax,%rax
xor    %r10,%r10
cmp    $0x0,%rdx
je     no_data
cmp    $0x10,%rdx
jne    loop
mov    (%rdi),%r10
mov    0x8(%rdi),%r11
mov    $0x1,%rdx
xor    (%rsi),%r10
xor    0x8(%rsi),%r11
or     %r11,%r10
cmovne %rdx,%rax
repz retq
loop:
mov    (%rdi),%r10b
lea    0x1(%rdi),%rdi
xor    (%rsi),%r10b
lea    0x1(%rsi),%rsi
or     %r10b,%al
dec    %rdx
jne    loop
neg    %rax
shr    $0x3f,%rax
no_data:
repz retq
```

Arithmetic DAG for all 3-byte inputs:



The power of modern reverse-engineering tools

Easy to use `angr.io` for automatic **symbolic execution**:
machine-language software ➜ arithmetic DAG.
Simplifies analysis: simpler instructions, no memory, no jumps.

The power of modern reverse-engineering tools

Easy to use `angr.io` for automatic **symbolic execution**:
machine-language software ➜ arithmetic DAG.
Simplifies analysis: simpler instructions, no memory, no jumps.

Limitation, sometimes exponential blowup: `angr` splits universes whenever it reaches an input-dependent branch or address.
. . . which we try to avoid in crypto anyway.

The power of modern reverse-engineering tools

Easy to use `angr.io` for automatic **symbolic execution**:
machine-language software ➜ arithmetic DAG.
Simplifies analysis: simpler instructions, no memory, no jumps.

Limitation, sometimes exponential blowup: `angr` splits universes whenever it reaches an input-dependent branch or address.
. . . which we try to avoid in crypto anyway.

`angr` (via Z3 SMT solver) often sees equivalence of small DAGs.
e.g. sees that OpenSSL `x86_64` `CRYPTO_memcmp` on 3-byte inputs outputs 0 if `x0==y0` and `x1==y1` and `x2==y2`,
and outputs 1 otherwise. Similarly for other input lengths.

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

unsigned char x[N];
unsigned char y[N];
int z;

int main()
{
  z = CRYPTO_memcmp(x,y,N);
  return 0;
}
```

```
#!/usr/bin/env python3

import sys
import angr

N = int(sys.argv[1]) if len(sys.argv) > 1 else 16

proj = angr.Project(’cmp%d’%N)
state = proj.factory.full_init_state()

state.options |= {
  angr.options.ZERO_FILL_UNCONSTRAINED_MEMORY
}
```

```
x = {}
xaddr = proj.loader.find_symbol(’x’).rebased_addr
for i in range(N):
  x[i] = state.solver.BVS(’x%d’%i,8)
  state.mem[xaddr+i].char = x[i]

y = {}
yaddr = proj.loader.find_symbol(’y’).rebased_addr
for i in range(N):
  y[i] = state.solver.BVS(’y%d’%i,8)
  state.mem[yaddr+i].char = y[i]

simgr = proj.factory.simgr(state)
simgr.run()
```

```
assert len(simgr.errored) == 0
print('%d universes' % len(simgr.deadended))
for exit in simgr.deadended:
  zaddr = proj.loader.find_symbol('z').rebased_addr
  z = exit.mem[zaddr].int.resolved
  print('out = %s' % z)

  xeqy = True
  for i in range(N):
    xeqy = state.solver.And(xeqy,x[i]==y[i])
  xney = state.solver.Not(xeqy)
  for bugs in ((z!=0,z!=1),(z!=0,xeqy),(z!=1,xney)):
    assert not exit.satisfiable(extra_constraints=bugs)
```

Symbolic execution with better equivalence testing

What if the DAG is too complicated for the SMT solver?
Answer: **Build smarter tools to recognize DAG equivalence.**

Symbolic execution with better equivalence testing

What if the DAG is too complicated for the SMT solver?
Answer: **Build smarter tools to recognize DAG equivalence.**

Case study, software library from `sorting.cr.yp.to`:

- New speed records for sorting of in-memory integer arrays.
  This is a subroutine in some post-quantum cryptosystems.
- Side-channel countermeasures:
  no secret branch conditions; no secret array indices.
- New tool verifies correct sorting of all size-N inputs.
  No need for manual review of per-CPU optimized code.