Crypto horror stories

Daniel J. Bernstein, Tanja Lange

University of Illinois at Chicago,
Ruhr University Bochum;
Eindhoven University of Technology
Mr Pichai said a combination of artificial intelligence and quantum would "help us tackle some of the biggest problems we see", but said it was important encryption evolved to match this.

"In a five to ten year time frame, quantum computing will break encryption as we know it today."

This is because current encryption methods, by which information such as texts or passwords is turned into code to make it unreadable, rely upon the fact that classic computers would take billions of years to decipher that code.

Quantum computers, with their ability to be..."
U.S. National Academy of Sciences report

Don’t panic. “Key Finding 1: Given the current state of quantum computing and recent rates of progress, it is highly unexpected that a quantum computer that can compromise RSA 2048 or comparable discrete logarithm-based public key cryptosystems will be built within the next decade.”
Don’t panic. “Key Finding 1: Given the current state of quantum computing and recent rates of progress, it is highly unexpected that a quantum computer that can compromise RSA 2048 or comparable discrete logarithm-based public key cryptosystems will be built within the next decade.”

Panic. “Key Finding 10: Even if a quantum computer that can decrypt current cryptographic ciphers is more than a decade off, the hazard of such a machine is high enough—and the time frame for transitioning to a new security protocol is sufficiently long and uncertain—that prioritization of the development, standardization, and deployment of post-quantum cryptography is critical for minimizing the chance of a potential security and privacy disaster.”
Many stages of research from design to deployment

Define the goals

Explore space of cryptosystems

Study algorithms for the attackers

Focus on secure cryptosystems

Study algorithms for the users

Study implementations on real hardware

Study side-channel attacks, fault attacks, etc.

Focus on secure, reliable implementations

Focus on implementations meeting performance requirements

Integrate securely into real-world applications

Crypto horror stories
Is post-quantum crypto moving quickly enough?


Is post-quantum crypto moving quickly enough?


Is post-quantum crypto moving quickly enough?


2014: EU solicits grant proposals in post-quantum crypto.

2014: ETSI starts working group on “Quantum-safe” crypto.

2017: Submissions to the NIST competition

21 December 2017: NIST posts 69 submissions from 260 people.

BIG QUAKE. BIKE. CFPKM. Classic McEliece. Compact LWE.
CRYSTALS-DILITHIUM. CRYSTALS-KYBER. DARGS. Ding Key Exchange.
DME. DRS. DualModeMS. Edon-K. EMBLEM and R.EMBLEM. FALCON.
HiMQ-3. HK17. HQC. KINDI. LAC. LAKE. LEDAkem. LEDApkc. Lepton.
LIMA. Lizard. LOCKER. LOTUS. LUOV. McNie. Mersenne-756839. MQDSS.
NewHope. NTRUEncrypt. pqNTRUSign. NTRU-HRSS-KEM. NTRU Prime.
pqRSA encryption. pqRSA signature. pqsigRM. QC-MDPC KEM. qTESLA.
RaCoSS. Rainbow. Ramstake. RankSign. RLCE-KEM. Round2. RQC. RVB.
SABER. SIKE. SPHINCS+. SRTPI. Three Bears. Titanium. WalnutDSA.

Some less secure than claimed; some smashed; some attack scripts.
Some submissions are broken within days

By end of 2017: 8 out of 69 submissions attacked.


Some less secure than claimed; some smashed; some attack scripts.
Do cryptographers have any idea what they’re doing?

By end of 2018: 22 out of 69 submissions attacked.


Some less secure than claimed; some smashed; some attack scripts.
Do cryptographers have any idea what they’re doing?

By end of 2019: 30 out of 69 submissions attacked.


Some less secure than claimed; some smashed; some attack scripts.
An attempt to explain the situation

People often categorize submissions. Examples of categories:

- Code-based encryption and signatures.
- Hash-based signatures.
- Isogeny-based encryption.
- Lattice-based encryption and signatures.
- Multivariate-quadratic encryption and signatures.
An attempt to explain the situation

“What’s safe is lattice-based cryptography.”
An attempt to explain the situation

“What’s safe is lattice-based cryptography.”

2006 Silverman: “Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography.”
An attempt to explain the situation

“What’s safe is lattice-based cryptography.”

2006 Silverman: “Lattices, SVP and CVP, have been intensively studied for more than 100 years, both as intrinsic mathematical problems and for applications in pure and applied mathematics, physics and cryptography.”

2017 Peikert: “The underlying worst-case problems—e.g., approximating short vectors in lattices—have been deeply studied by some of the great mathematicians and computer scientists going back at least to Gauss, and appear to be very hard.”
Reality: SVP hardness is poorly understood

Best SVP algorithms known by 2000:
- time $2^{\Theta(N \log N)}$ for almost all dimension-$N$ lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$. Huge change!

Approximate $c$ for some algorithms believed to take time $2^{\Theta(N)} + o(1)$:
- $0.415$: 2008 Nguyen–Vidick.
- $0.415$: 2010 Micciancio–Voulgaris.
- $0.378$: 2013 Zhang–Pan–Hu.
- $0.337$: 2014 Laarhoven.
- $0.298$: 2015 Laarhoven–de Weger.
Reality: SVP hardness is poorly understood

Best SVP algorithms known by 2000:
time $2^{\Theta(N \log N)}$ for almost all dimension-$N$ lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$. Huge change!

Approximate $c$ for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
Reality: SVP hardness is poorly understood

Best SVP algorithms known by 2000:
time $2^{\Theta(N \log N)}$ for almost all dimension-$N$ lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$. Huge change!

Approximate $c$ for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
Reality: SVP hardness is poorly understood

Best SVP algorithms known by 2000:
time $2^{\Theta(N \log N)}$ for almost all dimension-$N$ lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$. Huge change!

Approximate $c$ for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.378: 2013 Zhang–Pan–Hu.
Reality: SVP hardness is poorly understood

Best SVP algorithms known by 2000:
time $2^{\Theta(N \log N)}$ for almost all dimension-$N$ lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$. Huge change!

Approximate $c$ for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
Reality: SVP hardness is poorly understood

Best SVP algorithms known by 2000: time $2^{\Theta(N \log N)}$ for almost all dimension-$N$ lattices.

Best SVP algorithms known today: $2^{\Theta(N)}$. Huge change!

Approximate $c$ for some algorithms believed to take time $2^{(c+o(1))N}$:
0.415: 2008 Nguyen–Vidick.
0.415: 2010 Micciancio–Voulgaris.
0.378: 2013 Zhang–Pan–Hu.
0.337: 2014 Laarhoven.
0.298: 2015 Laarhoven–de Weger.
Lattice security is even more poorly understood

Lattice-based crypto has many more attack avenues than SVP.

Lattice security is even more poorly understood

Lattice-based crypto has many more attack avenues than SVP.


Lattice security estimates are so imprecise that nobody is sure whether the remaining submissions are damaged by a 2019 paper solving a lattice problem “more than a million times faster”.
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.
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.
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

Register article
ELLIPITISCHE KURVEN

Minerva-Angriff zielt auf zertifizierte Krypto-Chips


4. Oktober 2019, 13:41 Uhr, Hanno Böck
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA,

Password is MUNICH.

1974: Exploit developed by Alan Bell for TENEX operating system.
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB,

Password is MUNICH.

1974: Exploit developed by Alan Bell for TENEX operating system.
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, ...
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.
Try MAA,
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.
Try MAA, MBB,
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.
Try MAA, MBB, MCC, . . .
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.  
Try MAA, MBB, MCC, . . . , MUU takes slightly longer to fail.  
Try MUA,
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.
Try MAA, MBB, MCC, . . . , MUU takes slightly longer to fail.
Try MUA, MUB,
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, ..., MMM takes slightly longer to fail.
Try MAA, MBB, MCC, ..., MUU takes slightly longer to fail.
Try MUA, MUB, MUC, ...
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.
Try MAA, MBB, MCC, . . . , MUU takes slightly longer to fail.
Try MUA, MUB, MUC, . . . , MUN takes slightly longer to fail.

Password is MUNICH.

1974: Exploit developed by Alan Bell for TENEX operating system.
Timing attacks are not a new phenomenon

Password recovery if server compares letter by letter:
Try AAA, BBB, CCC, . . . , MMM takes slightly longer to fail.
Try MAA, MBB, MCC, . . . , MUU takes slightly longer to fail.
Try MUA, MUB, MUC, . . . , MUN takes slightly longer to fail.

Password is MUNICH.

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):
    $m = m^2 \mod n$
    if $D[i] == 1$:
        $m = m \times c \mod n$

print($m$)
Exponentiation with secret exponent (RSA, DH)

Compute $c^d$ given $c$ and $d$.

\begin{align*}
n &= 1000001 \\
d &= 12473 \\
c &= 41241 \\
l &= d.nbits() \\
D &= d.bits() \\
m &= c \\
\text{for } i \text{ in range}(l-2,-1,-1): \quad \# \text{ loop length depends on } d \\
&\quad m = m^2 \mod n \\
&\quad \text{if } D[i] == 1: \\
&\quad \quad m = m \times c \mod n \\
\text{print}(m)
\end{align*}
Exponentiation with secret exponent (RSA, DH)

Compute $c^d$ given $c$ and $d$.

\[
\begin{align*}
n &= 1000001 \\
d &= 12473 \\
c &= 41241 \\
l &= d.\text{nbits}() \\
D &= d.\text{bits}() \\
m &= c \\
\text{for } i \text{ in range}(l-2,-1,-1): \quad \# \text{ loop length depends on } d \\
\quad &\quad m = m^2 \mod n \\
\quad &\quad \text{if } D[i] == 1: \quad \# \text{ branch depends on } d \\
\quad \quad &\quad m = m \times c \mod n \\
\text{print}(m)
\end{align*}
\]
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: $14019 = \cdots$
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: \(14019 = 0x36C3 = 0011\ 0110\ 1100\ 0011\)

Precompute \(c_1, c^2,\) and \(c^3\). \(c^{14019} = (((((c^3)^4 \cdot (c^4)^4 \cdot (c^2)^4 \cdot (c^3)^4) \cdot (c^3)^4))^4 \cdot (c^3)^4)\). The same number of squarings, 4 instead of 7 multiplications.

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: $14019 = 0x36C3 = 0011 \ 0110 \ 1100 \ 0011$

$c_{14019} = (((c_3 \cdot c_4) \cdot c_4) \cdot c_3) \cdot \ldots \cdot c_3$. Same number of squarings, 4 instead of 7 multiplications.
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: $14019 = 0x36C3 = 0011 0110 1100 0011$

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

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

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.

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.
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

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

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 $\infty$, so initialization at $\infty$ 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.
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 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.
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.

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.
#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 Julian Seward et al.
==2599== Using Valgrind-3.14.0 and LibVEX; rerun with -h for help
==2599== Command: ./rc4
==2599==
==2599== Use of uninitialised value of size 8
==2599== at 0x4A1A0EF: RC4_set_key (in /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1)
==2599== by 0x1090BD: main (in /home/.../rc4)
...
==2599== ERROR SUMMARY: 256 errors from 1 contexts (suppressed: 0 from 0)
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.
Now we have constant-time exponentiation / scalar multiplication if

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

Single-clock-cycle instructions are probably OK.
### Table 18-1 Instruction timings (continued)

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>Size</th>
<th>Cycles count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift operations</td>
<td>32</td>
<td>1</td>
<td>ASR{S}, LSL{S}, LSR{S}, ROR{S}, and RRX{S}.</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>32</td>
<td>1</td>
<td>REV, REVH, REVSH, RBIT, CLZ, SXTB, SXTH, UXTB, and UXTH. Extension instructions same as corresponding ARMv6 16-bit instructions.</td>
</tr>
<tr>
<td>Table Branch</td>
<td>16</td>
<td>4+P&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Table branches for switch/case use. These are LDR, shifts and then branch.</td>
</tr>
<tr>
<td>Multiply</td>
<td>32</td>
<td>1 or 2</td>
<td>MUL, MLA, and MLS. MUL is one cycle and MLS and MLA are two cycles.</td>
</tr>
<tr>
<td>Multiply with 64-bit result</td>
<td>32</td>
<td>3-7&lt;sup&gt;c&lt;/sup&gt;</td>
<td>UMULL, SMULL, UMLAL, and SMLAL. Cycle count based on input sizes. That is, ABS(inputs) &lt; 64K terminates early.</td>
</tr>
<tr>
<td>Load-store addressing</td>
<td>32</td>
<td>-</td>
<td>Supports Format PC+/-imm12, Rbase+imm12, Rbase+/-imm8, and adjusted register including shifts. T variants used when in Privilege mode.</td>
</tr>
</tbody>
</table>

---

<sup>a</sup> 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.
Flow chart for UMLAL (unsigned multiply add) from A performance study of X25519 on Cortex-M3 and M4 by Wouter de Groot.
This #PatchTuesday you are strongly encouraged to implement the recently released CVE-2020-0601 patch immediately.

media.defense.gov/2020/Jan/14/20...
Microsoft CVE-2020-0601

- Certificate shows Alice’s public key $Q$ and params $E, P$.
- Signed message consists of ECDSA signature $(m, r, s)$ as well as Alice’s public key $Q$ and $E, P$. After checking certificate, Windows remembers that $Q$ is Alice’s trusted public key.
- Next verification of a signature by Alice checks validity of $(m', r', s')$ under supplied $(E, P, Q)$ if $Q$ is in database.
Certificate shows Alice’s public key $Q$ and params $E, P$.

Signed message consists of ECDSA signature $(m, r, s)$ as well as Alice’s public key $Q$ and $E, P$. After checking certificate, Windows remembers that $Q$ is Alice’s trusted public key.

Next verification of a signature by Alice checks validity of $(m', r', s')$ under supplied $(E', P', Q)$ if $Q$ is in database.
Microsoft CVE-2020-0601

- Certificate shows Alice’s public key $Q$ and params $E, P$.
- Signed message consists of ECDSA signature $(m, r, s)$ as well as Alice’s public key $Q$ and $E, P$. After checking certificate, Windows remembers that $Q$ is Alice’s trusted public key.
- Next verification of a signature by Alice checks validity of $(m', r', s')$ under supplied $(E', P', Q)$ if $Q$ is in database.
- Easiest attack: use $E' = E$, $P' = Q$, secret key 1.
Microsoft CVE-2020-0601

- Certificate shows Alice’s public key \( Q \) and params \( E, P \).
- Signed message consists of ECDSA signature \((m, r, s)\) as well as Alice’s public key \( Q \) and \( E, P \). After checking certificate, Windows remembers that \( Q \) is Alice’s trusted public key.
- Next verification of a signature by Alice checks validity of \((m', r', s')\) under supplied \((E', P', Q)\) if \( Q \) is in database.
- Easiest attack: use \( E' = E, P' = Q \), secret key 1.
- Vaudenay (2004) Digital Signature Schemes with Domain Parameters (thanks to Cas Cremers for reference)
- Landed in Windows code base in a move to support arbitrary elliptic curves
Microsoft CVE-2020-0601

- Certificate shows Alice’s public key $Q$ and params $E, P$.
- Signed message consists of ECDSA signature $(m, r, s)$ as well as Alice’s public key $Q$ and $E, P$. After checking certificate, Windows remembers that $Q$ is Alice’s trusted public key.
- Next verification of a signature by Alice checks validity of $(m', r', s')$ under supplied $(E', P', Q)$ if $Q$ is in database.
- Easiest attack: use $E' = E$, $P' = Q$, secret key 1.
- Vaudenay (2004) Digital Signature Schemes with Domain Parameters (thanks to Cas Cremers for reference)
- Landed in Windows code base in a move to support arbitrary elliptic curves . . . in 2015.
Microsoft CVE-2020-0601

Daniel J. Bernstein
@hashbreaker

Replying to @hanno

See cr.yp.to/newelliptic/ni... (from @hyperelliptic and me), which says in §1 that "unnecessary complexity in ECC implementations" creates "ECC security failures", and says in §11 that allowing run-time curve choices causes "obvious damage to implementation simplicity". Told ya so.

8:35 PM · Jan 15, 2020 · Twitter Web App
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?
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.”

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     $0x1fffffffff, %eax
    imulq   16-128($ap),%rbx
@@ -1329,15 +1331,12 @@
September 2019: bug announced in Falcon software


“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 \(\Rightarrow\) 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:

\[\vdash \alpha, \beta \in 1. C : \alpha \land \beta = \Lambda. \equiv. \alpha \lor \beta \in 2\]

**Dem.**

\[\vdash \ast 54.26. C \vdash : \alpha = \iota x. \beta = \iota y. C : \alpha \lor \beta \in 2. \equiv. x \neq y.\]

\[\ast 51.231\]

\[\ast 13.12\]

\[\vdash (1). \ast 11.11.35. C\]

\[\vdash (\exists x, y). \alpha = \iota x. \beta = \iota y. C : \alpha \lor \beta \in 2. \equiv. \alpha \land \beta = \Lambda\]  

(1)

\[\vdash (2). \ast 11.54. \ast 52.1. C \vdash . \text{Prop}\]

\[\vdash (\exists x, y). \alpha = \iota x. \beta = \iota y. C : \alpha \lor \beta \in 2. \equiv. \alpha \land \beta = \Lambda\]  

(2)

From this proposition it will follow, when arithmetical addition has been defined, that \(1 + 1 = 2\).
Formal verification today

Require code reviewer to \textit{prove} correctness.
Require proofs to pass a proof-checking tool.
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.
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_mempcmp 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.”
Can we fix this?
Symbolic testing: beyond testing particular inputs

```
.globl CRYPTO_memcmp
CRYPTO_memcmp:
xor  %rax,%rax
xor  %r10,%r10
cmp  $0x0,%rdx
ej  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 $\rightarrow$ 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](http://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.