Sorting integer arrays: security, speed, and verification

D. J. Bernstein

Bob's laptop screen:

```
From: Alice


Thank you for your
submission. We received
many interesting papers,
and unfortunately your
```

Bob assumes this message is something Alice actually sent.

But today's "security" systems fail to guarantee this property. Attacker could have modified or forged the message.

Trusted computing base (TCB)

TCB: portion of computer system that is responsible for enforcing the users' security policy.

Trusted computing base (TCB)

TCB: portion of computer system that is responsible for enforcing the users' security policy.

Security policy for this talk:
If message is displayed on Bob's screen as "`From: Alice`" then message is from Alice.

Trusted computing base (TCB)

TCB: portion of computer system that is responsible for enforcing the users' security policy.

Security policy for this talk:
If message is displayed on Bob's screen as "`From: Alice`" then message is from Alice.

If TCB works correctly, then message is guaranteed to be from Alice, no matter what the rest of the system does.

Examples of attack strategies:

1. Attacker uses buffer overflow in a device driver to control Linux kernel on Alice's laptop.

Examples of attack strategies:

1. Attacker uses buffer overflow in a device driver to control Linux kernel on Alice's laptop.
2. Attacker uses buffer overflow in a web browser to control disk files on Bob's laptop.

Examples of attack strategies:

1. Attacker uses buffer overflow in a device driver to control Linux kernel on Alice's laptop.

2. Attacker uses buffer overflow in a web browser to control disk files on Bob's laptop.

Device driver is in the TCB.
Web browser is in the TCB.
CPU is in the TCB. Etc.

Examples of attack strategies:

1. Attacker uses buffer overflow in a device driver to control Linux kernel on Alice's laptop.

2. Attacker uses buffer overflow in a web browser to control disk files on Bob's laptop.

Device driver is in the TCB.
Web browser is in the TCB.
CPU is in the TCB. Etc.

Massive TCB has many bugs, including many security holes.
Any hope of fixing this?

Classic security strategy:

Rearchitect computer systems
to have a much smaller TCB.

Classic security strategy:

Rearchitect computer systems
to have a much smaller TCB.

Carefully audit the TCB.

Classic security strategy:

Rearchitect computer systems to have a much smaller TCB.

Carefully audit the TCB.

e.g. Bob runs many VMs:

VM A Alice data	VM C Charlie data	. . .

TCB stops each VM from touching data in other VMs.

Classic security strategy:

Rearchitect computer systems
to have a much smaller TCB.

Carefully audit the TCB.

e.g. Bob runs many VMs:

VM A Alice data	VM C Charlie data	. . .

TCB stops each VM from
touching data in other VMs.

Browser in VM C isn't in TCB.
Can't touch data in VM A,
if TCB works correctly.

Classic security strategy:

Rearchitect computer systems
to have a much smaller TCB.

Carefully audit the TCB.

e.g. Bob runs many VMs:

VM A	VM C	
Alice data	Charlie data	. . .

TCB stops each VM from
touching data in other VMs.

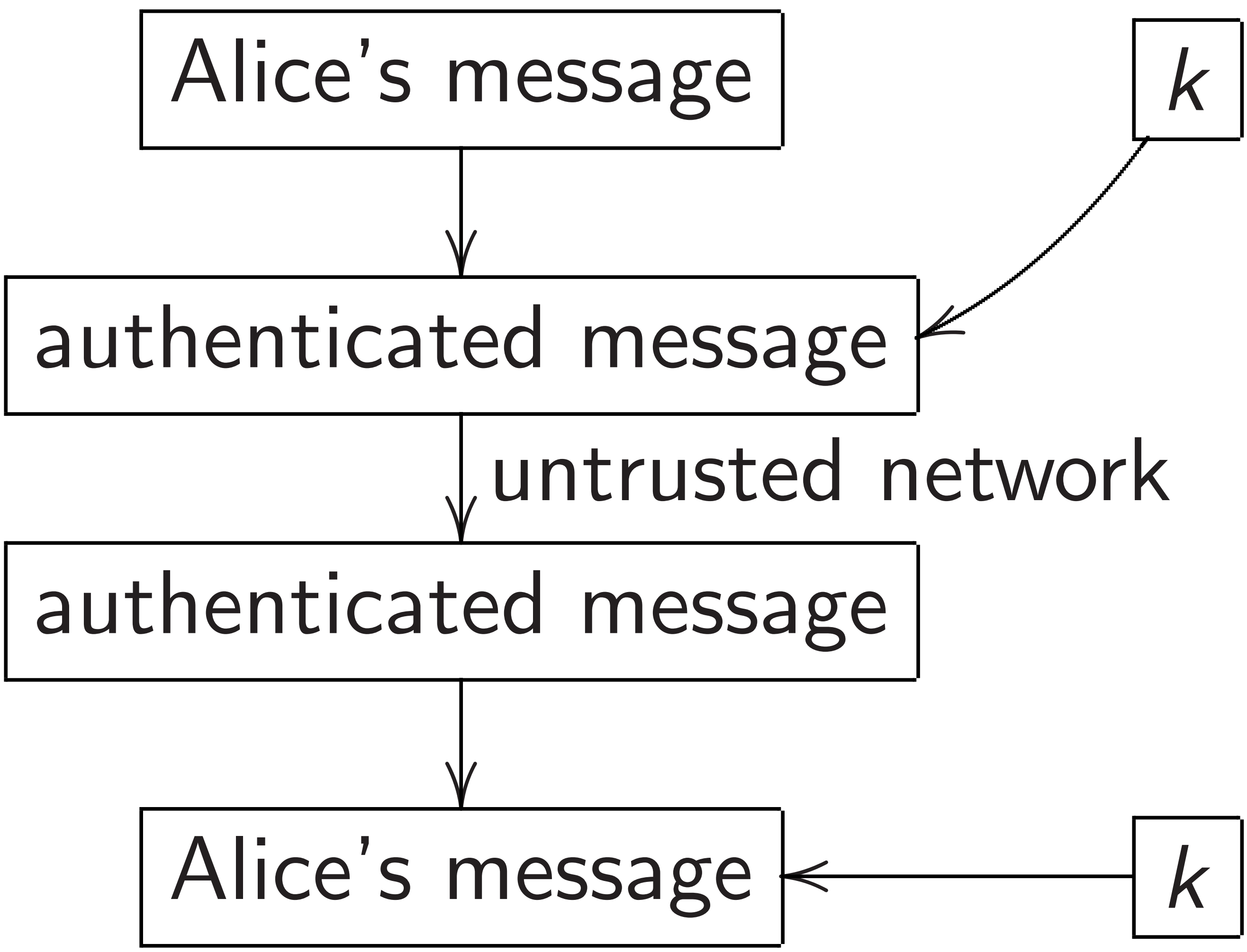
Browser in VM C isn't in TCB.
Can't touch data in VM A,
if TCB works correctly.

Alice also runs many VMs.

Cryptography

How does Bob's laptop know that incoming network data is from Alice's laptop?

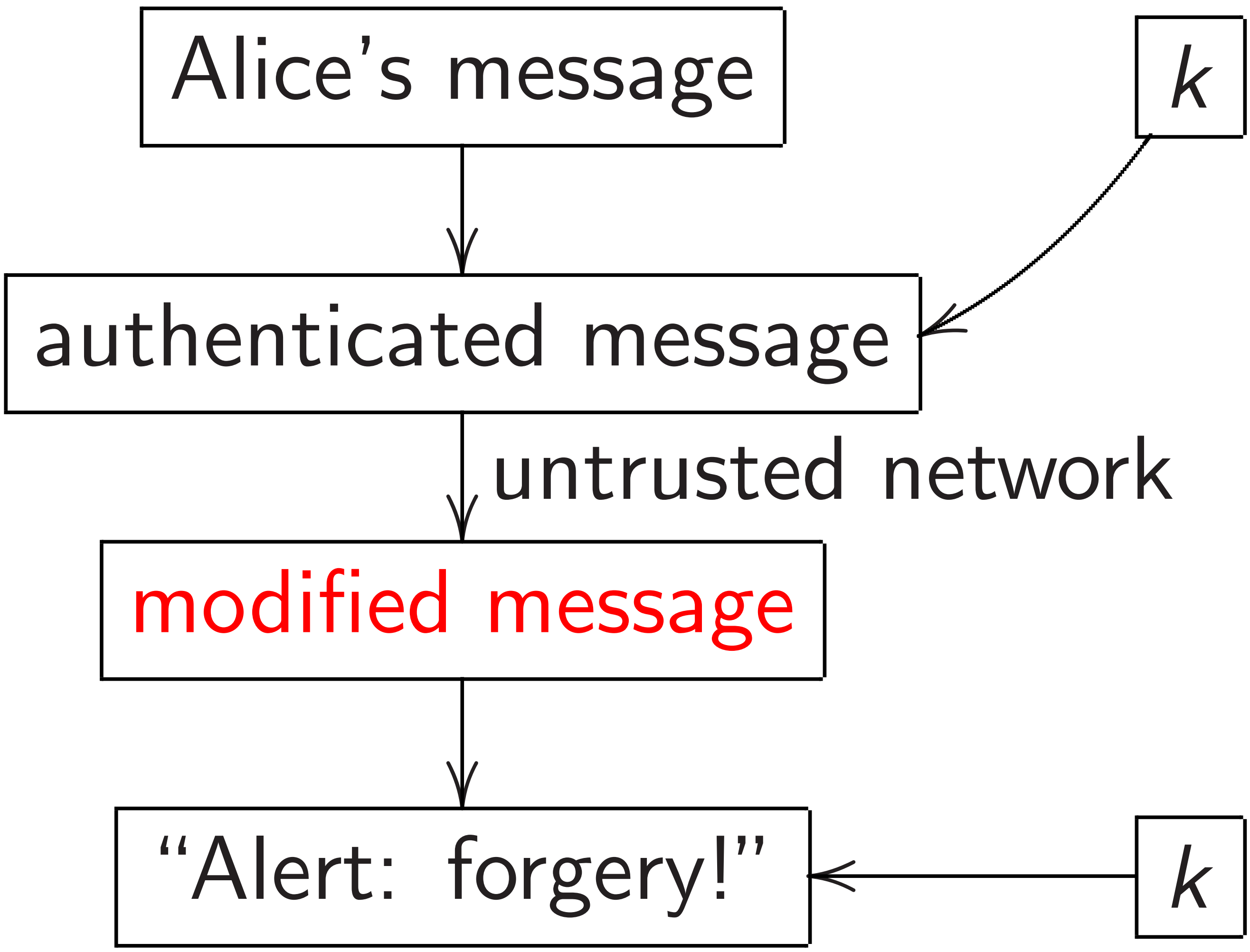
Cryptographic solution: Message-authentication codes.



Cryptography

How does Bob's laptop know that incoming network data is from Alice's laptop?

Cryptographic solution: Message-authentication codes.



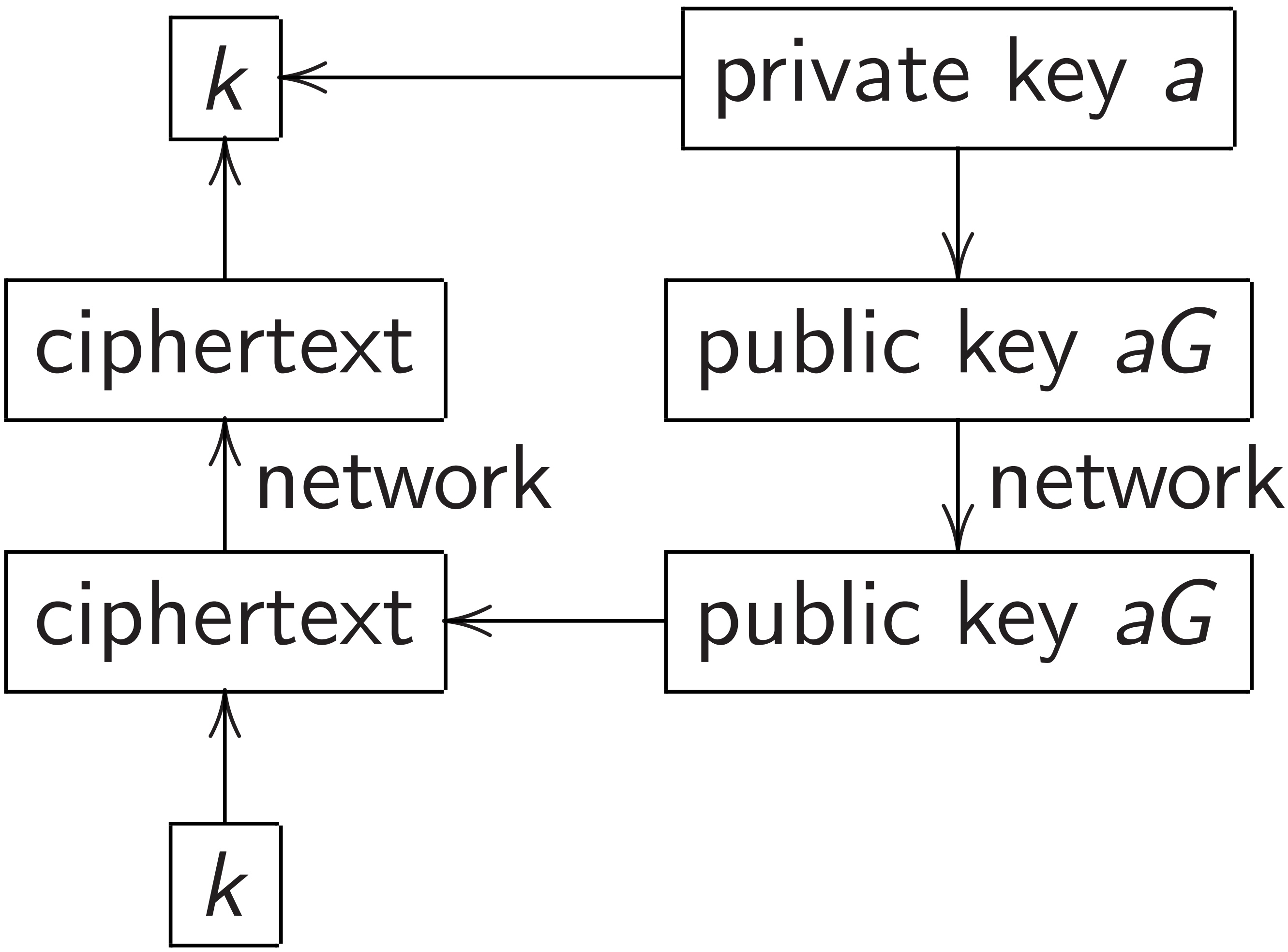
Important for Alice and Bob to share the same secret k.

What if attacker was spying on their communication of k?

Important for Alice and Bob to share the same secret k.

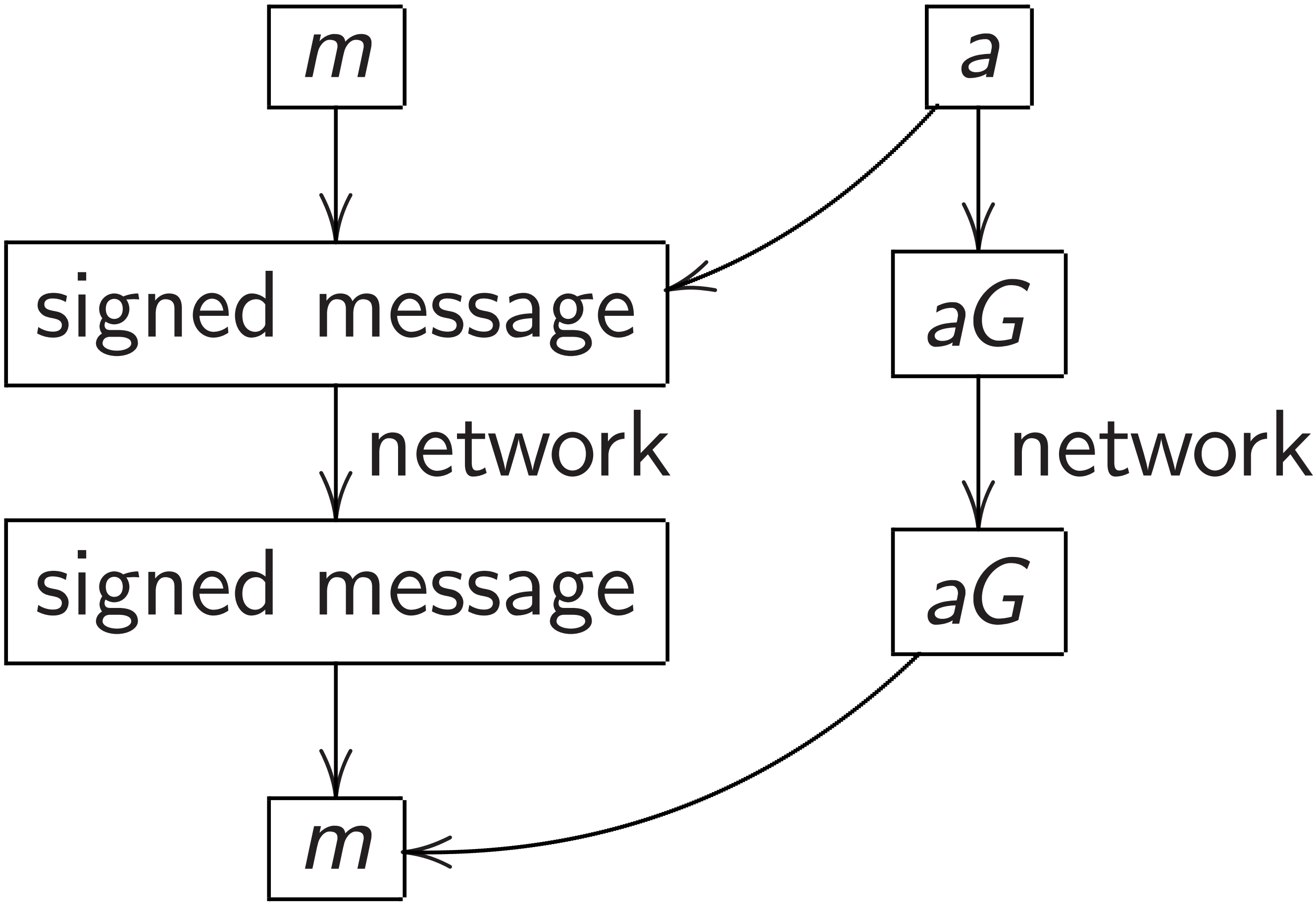
What if attacker was spying on their communication of k?

Solution 1:
Public-key encryption.



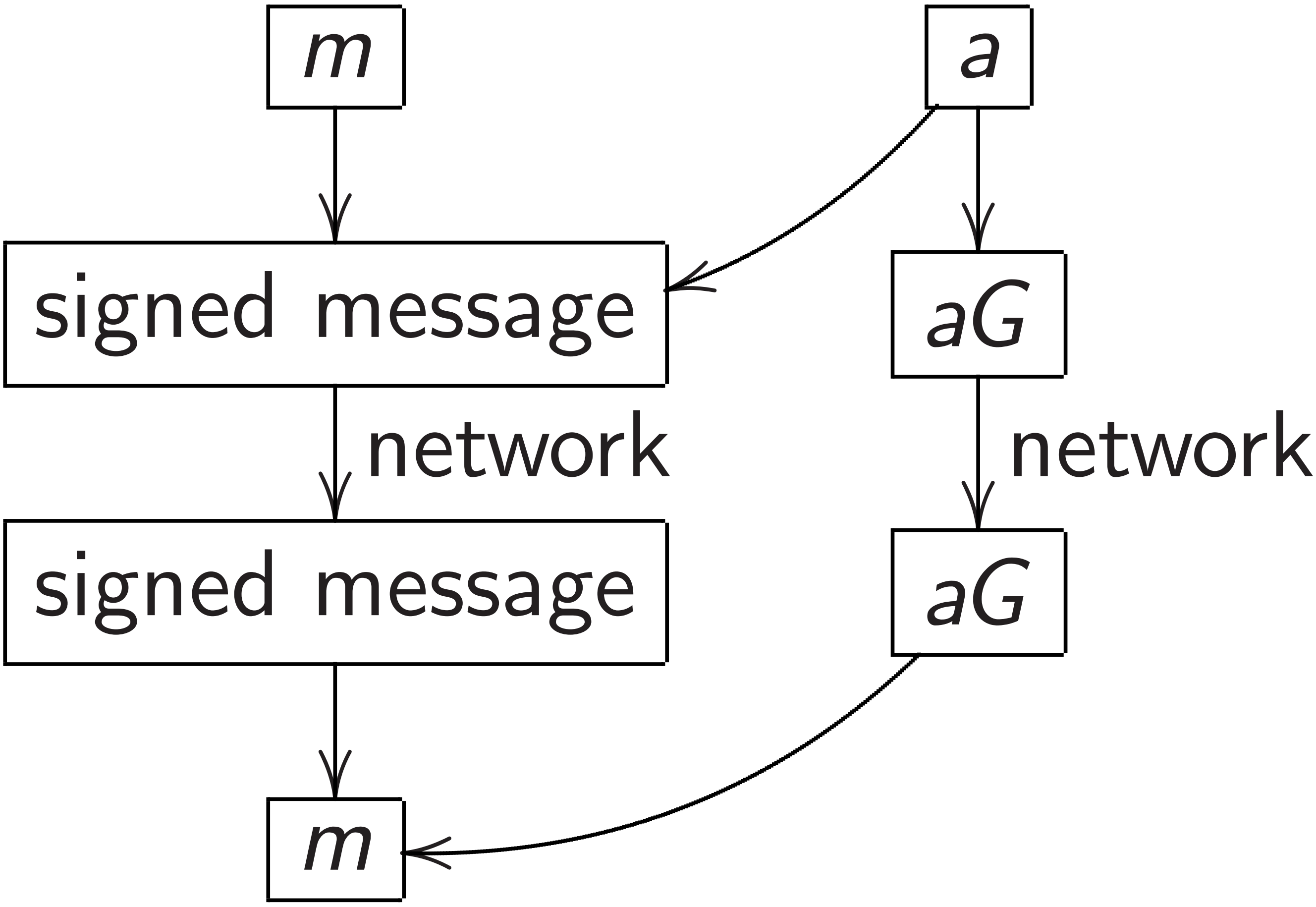
Solution 2:

Public-key signatures.



Solution 2:

Public-key signatures.



No more shared secret k

but Alice still has secret a.

Cryptography requires TCB

to protect secrecy of keys,

even if user has no other secrets.

Constant-time software

Large portion of CPU hardware:
optimizations depending on
addresses of memory locations.

Consider data caching,
instruction caching,
parallel cache banks,
store-to-load forwarding,
branch prediction, etc.

Constant-time software

Large portion of CPU hardware: optimizations depending on addresses of memory locations.

Consider data caching, instruction caching, parallel cache banks, store-to-load forwarding, branch prediction, etc.

Many attacks (e.g. TLBleed from 2018 Gras–Razavi–Bos–Giuffrida) show that this portion of the CPU has trouble keeping secrets.

Typical literature on this topic:

Understand this portion of CPU. But details are often proprietary, not exposed to security review.

Try to push attacks further. This becomes very complicated.

Tweak the attacked software to try to stop the known attacks.

Typical literature on this topic:

Understand this portion of CPU.
But details are often proprietary,
not exposed to security review.

Try to push attacks further.
This becomes very complicated.

Tweak the attacked software
to try to stop the known attacks.

For researchers: This is great!

Typical literature on this topic:

Understand this portion of CPU.
But details are often proprietary,
not exposed to security review.

Try to push attacks further.
This becomes very complicated.

Tweak the attacked software
to try to stop the known attacks.

For researchers: This is great!

For auditors: This is a nightmare.
Many years of security failures.
No confidence in future security.

The “constant-time” solution:
Don’t give any secrets
to this portion of the CPU.
(1987 Goldreich, 1990 Ostrovsky:
Oblivious RAM; 2004 Bernstein:
domain-specific for better speed)

The "constant-time" solution:
Don't give any secrets
to this portion of the CPU.
(1987 Goldreich, 1990 Ostrovsky:
Oblivious RAM; 2004 Bernstein:
domain-specific for better speed)

TCB analysis: Need this portion
of the CPU to be correct, but
don't need it to keep secrets.
Makes auditing much easier.

The "constant-time" solution:
Don't give any secrets
to this portion of the CPU.
(1987 Goldreich, 1990 Ostrovsky:
Oblivious RAM; 2004 Bernstein:
domain-specific for better speed)

TCB analysis: Need this portion
of the CPU to be correct, but
don't need it to keep secrets.
Makes auditing much easier.

Good match for attitude and
experience of CPU designers: e.g.,
Intel issues errata for correctness
bugs, not for information leaks.

Case study: Constant-time sorting

Serious risk within 10 years:
Attacker has quantum computer
breaking today's most popular
public-key crypto (RSA and ECC;
e.g., finding a given aG).

Case study: Constant-time sorting

Serious risk within 10 years: Attacker has quantum computer breaking today's most popular public-key crypto (RSA and ECC; e.g., finding a given aG).

2017: Hundreds of people submit 69 complete proposals to international competition for post-quantum crypto standards.

Case study: Constant-time sorting

Serious risk within 10 years: Attacker has quantum computer breaking today's most popular public-key crypto (RSA and ECC; e.g., finding a given aG).

2017: Hundreds of people submit 69 complete proposals to international competition for post-quantum crypto standards.

Subroutine in some submissions: sort array of secret integers. e.g. sort 768 32-bit integers.

How to sort secret data without any secret addresses?

How to sort secret data without any secret addresses?

Typical sorting algorithms—merge sort, quicksort, etc.—choose load/store addresses based on secret data. Usually also branch based on secret data.

How to sort secret data
without any secret addresses?

Typical sorting algorithms—
merge sort, quicksort, etc.—
choose load/store addresses
based on secret data. Usually
also branch based on secret data.

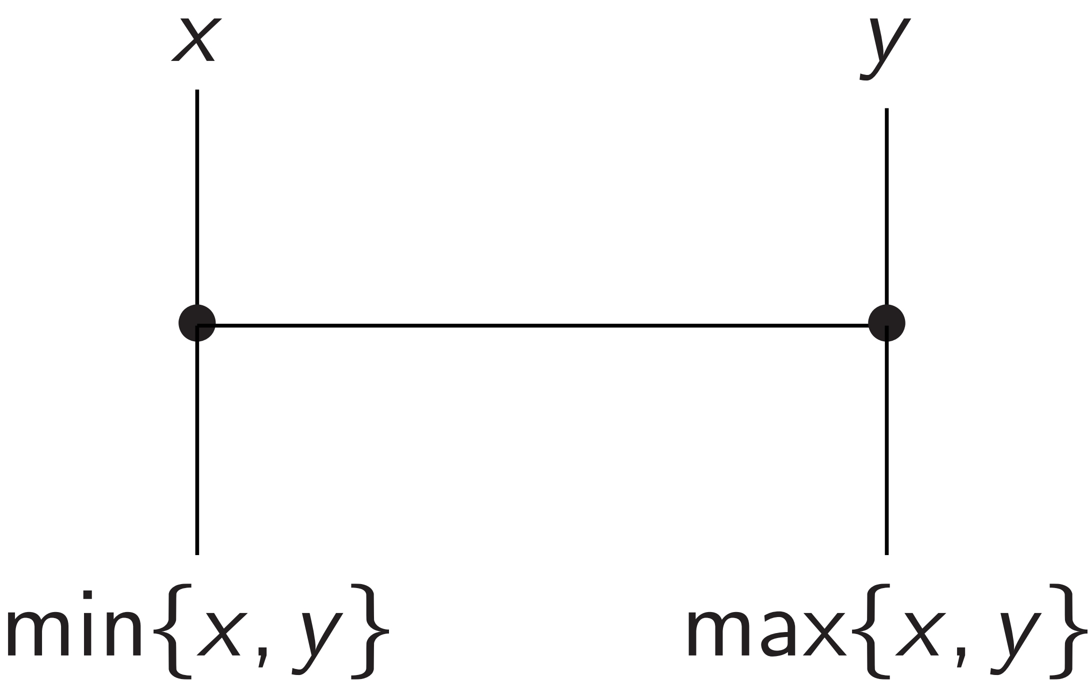
One submission to competition:
"Radix sort is used as
constant-time sorting algorithm."
Some versions of radix sort
avoid secret branches.

How to sort secret data without any secret addresses?

Typical sorting algorithms—merge sort, quicksort, etc.—choose load/store addresses based on secret data. Usually also branch based on secret data.

One submission to competition: "Radix sort is used as constant-time sorting algorithm." Some versions of radix sort avoid secret branches. But data addresses in radix sort still depend on secrets.

Foundation of solution:
a **comparator** sorting 2 integers.

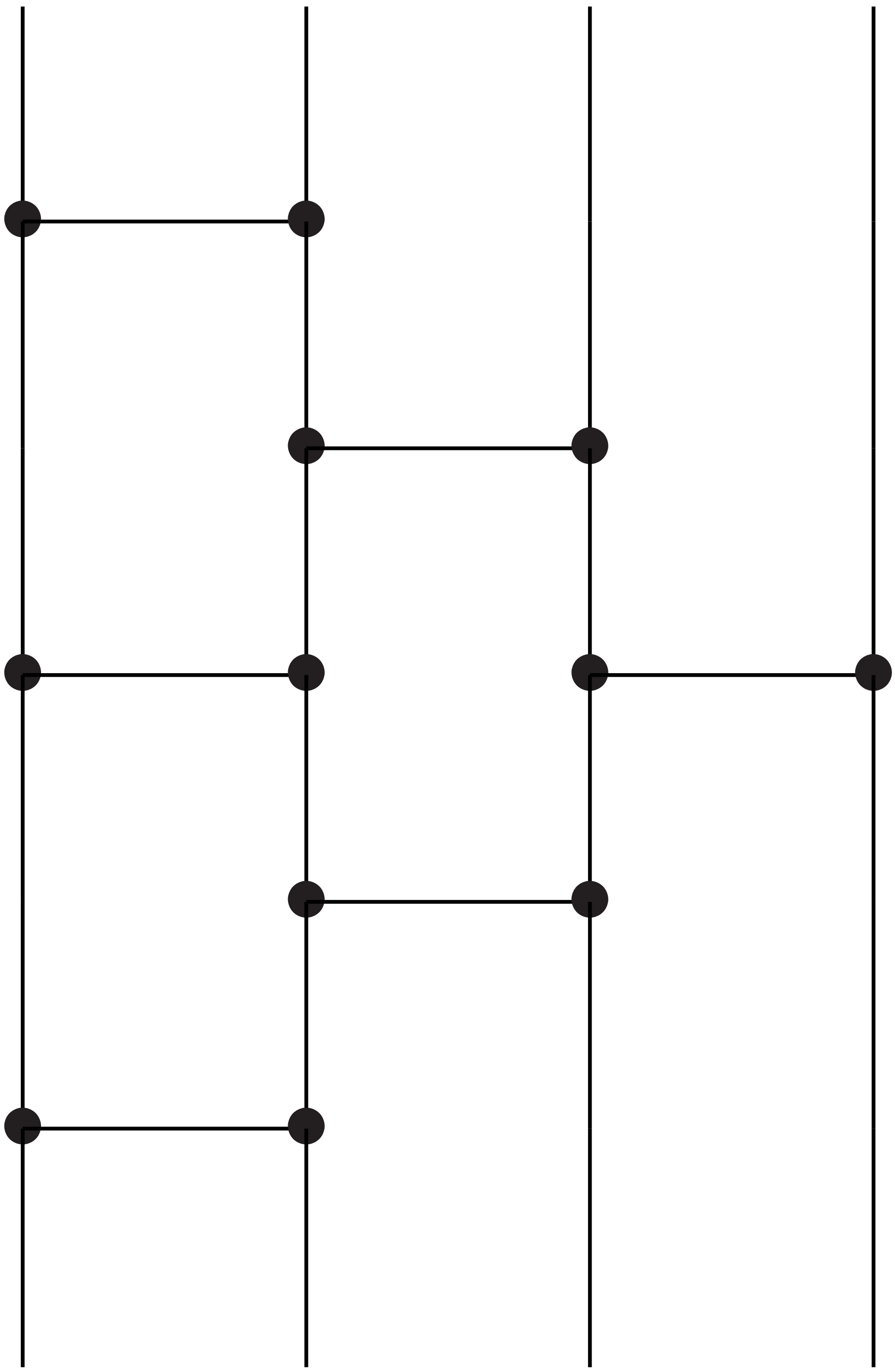


Easy constant-time exercise in C.
Warning: C standard allows compiler to screw this up.

Even easier exercise in asm.

Combine comparators into a **sorting network** for more inputs.

Example of a sorting network:



Positions of comparators
in a sorting network are
independent of the input.
Naturally constant-time.

Positions of comparators in a sorting network are independent of the input. Naturally constant-time.

But $(n^2 - n)/2$ comparators produce complaints about performance as n increases.

Positions of comparators in a sorting network are independent of the input. Naturally constant-time.

But $(n^2 - n)/2$ comparators produce complaints about performance as n increases.

Speed is a serious issue in the post-quantum competition. "Cost" is evaluation criterion; "we'd like to stress this once again on the forum that we'd really like to see more platform-optimized implementations"; etc.

```
void int32_sort(int32 *x,int64 n)
{ int64 t,p,q,i;
  if (n < 2) return;
  t = 1;
  while (t < n - t) t += t;
  for (p = t;p > 0;p >>= 1) {
    for (i = 0;i < n - p;++i)
      if (!(i & p))
        minmax(x+i,x+i+p);
    for (q = t;q > p;q >>= 1)
      for (i = 0;i < n - q;++i)
        if (!(i & p))
          minmax(x+i+p,x+i+q);
  }
}
```

Previous slide: C translation of 1973 Knuth "merge exchange", which is a simplified version of 1968 Batcher "odd-even merge" sorting networks.

$\approx n(\log_2 n)^2/4$ comparators.
Much faster than bubble sort.

Warning: many other descriptions of Batcher's sorting networks require n to be a power of 2.
Also, Wikipedia says "Sorting networks . . . are not capable of handling arbitrarily large inputs."

This constant-time sorting code

↓ vectorization (for Haswell)

Constant-time sorting code
included in 2017
Bernstein–Chuengsatiansup–
Lange–van Vredendaal
“NTRU Prime” software release

↓ revamped for higher speed

New: “djbsort”
constant-time sorting code

The slowdown for constant time

Massive fast-sorting literature.

2015 Gueron–Krasnov: AVX and AVX2 (Haswell) optimization of quicksort. For 32-bit integers: $\approx$45 cycles/byte for $n \approx 2^{10}$, $\approx$55 cycles/byte for $n \approx 2^{20}$.

Slower than "the radix sort implemented of IPP, which is the fastest in-memory sort we are aware of": 32, 40 cycles/byte.

IPP: Intel's Integrated Performance Primitives library.

Constant-time results,
again on Haswell CPU core:

Constant-time results,
again on Haswell CPU core:

2017 BCLvV:
6.5 cycles/byte for $n \approx 2^{10}$,
33 cycles/byte for $n \approx 2^{20}$.

Constant-time results,
again on Haswell CPU core:

2017 BCLvV:
6.5 cycles/byte for $n \approx 2^{10}$,
33 cycles/byte for $n \approx 2^{20}$.

2018 djbsort:
2.5 cycles/byte for $n \approx 2^{10}$,
15.5 cycles/byte for $n \approx 2^{20}$.

Constant-time results,
again on Haswell CPU core:

2017 BCLvV:
6.5 cycles/byte for $n \approx 2^{10}$,
33 cycles/byte for $n \approx 2^{20}$.

2018 djbsort:
2.5 cycles/byte for $n \approx 2^{10}$,
15.5 cycles/byte for $n \approx 2^{20}$.

No slowdown. New speed records!

Constant-time results,
again on Haswell CPU core:

2017 BCLvV:
6.5 cycles/byte for $n \approx 2^{10}$,
33 cycles/byte for $n \approx 2^{20}$.

2018 djbsort:
2.5 cycles/byte for $n \approx 2^{10}$,
15.5 cycles/byte for $n \approx 2^{20}$.

No slowdown. New speed records!

Warning: Comparison for $n \approx 2^{20}$ involves microarchitecture details beyond Haswell core. Should measure all code on same CPU.

How can an $n(\log n)^2$ algorithm beat standard $n \log n$ algorithms?

How can an $n(\log n)^2$ algorithm beat standard $n \log n$ algorithms?

Answer: well-known trends in CPU design, reflecting fundamental hardware costs of various operations.

How can an $n(\log n)^2$ algorithm beat standard $n \log n$ algorithms?

Answer: well-known trends in CPU design, reflecting fundamental hardware costs of various operations.

Every cycle, Haswell core can do 8 "min" ops on 32-bit integers + 8 "max" ops on 32-bit integers.

How can an $n(\log n)^2$ algorithm beat standard $n \log n$ algorithms?

Answer: well-known trends in CPU design, reflecting fundamental hardware costs of various operations.

Every cycle, Haswell core can do 8 "min" ops on 32-bit integers + 8 "max" ops on 32-bit integers.

Loading a 32-bit integer from a random address: much slower.

Conditional branch: much slower.

Verification

Sorting software is in the TCB.
Does it work correctly?

Test the sorting software on many random inputs, increasing inputs, decreasing inputs. Seems to work.

Verification

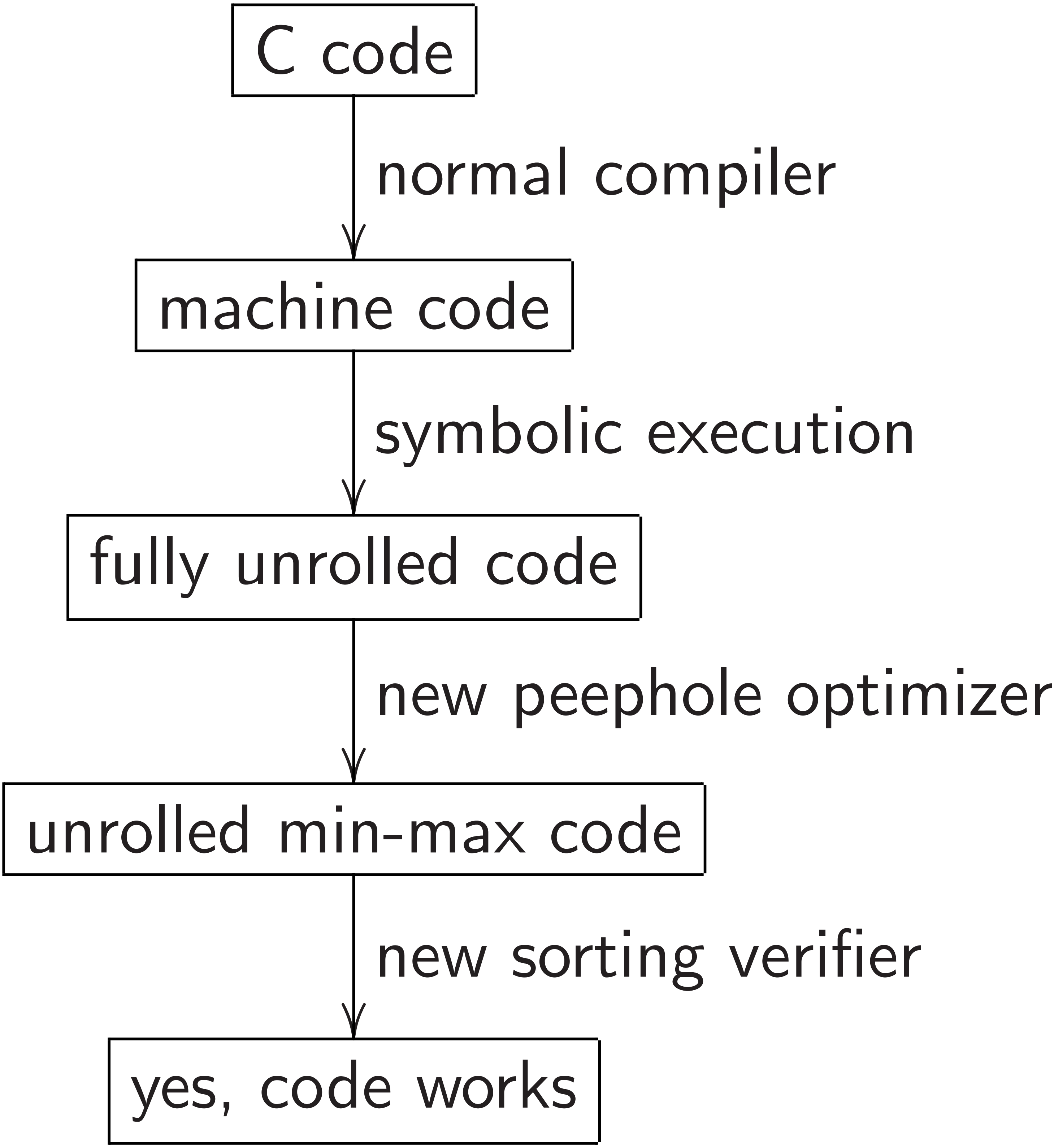
Sorting software is in the TCB.
Does it work correctly?

Test the sorting software on many random inputs, increasing inputs, decreasing inputs. Seems to work.

But are there *occasional* inputs where this sorting software fails to sort correctly?

History: Many security problems involve occasional inputs where TCB works incorrectly.

For each used n (e.g., 768):



Symbolic execution:
use existing “angr” library,
with tiny new patches for
eliminating byte splitting, adding
a few missing vector instructions.

Symbolic execution:
use existing “angr” library,
with tiny new patches for
eliminating byte splitting, adding
a few missing vector instructions.

Peephole optimizer:
recognize instruction patterns
equivalent to min, max.

Symbolic execution:
use existing “angr” library,
with tiny new patches for
eliminating byte splitting, adding
a few missing vector instructions.

Peephole optimizer:
recognize instruction patterns
equivalent to min, max.

Sorting verifier: decompose
DAG into merging networks.
Verify each merging network
using generalization of 2007
Even–Levi–Litman, correction of
1990 Chung–Ravikumar.

First djbsort release,
verified `int32` on AVX2:

https://sorting.cr.yp.to

Includes the sorting code;
automatic build-time tests;
simple benchmarking program;
verification tools.

Web site shows how to
use the verification tools.

Next release planned:
verified ARM NEON code
and verified portable code.