CryptAttackTester: high-assurance attack analysis

Daniel J. Bernstein, Tung Chou

For Crypto 2024 paper and software:

https://cat.cr.yp.to

An incorrect analysis of a factorization algorithm

1984 Schnorr–Lenstra claimed that "every composite integer n will be factored in $o(\exp\sqrt{\ln n \ln\ln n})$ bit operations" by the algorithm in that paper.

An incorrect analysis of a factorization algorithm

1984 Schnorr–Lenstra claimed that "every composite integer n will be factored in $o(\exp \sqrt{\ln n \ln \ln n})$ bit operations" by the algorithm in that paper.

1992 Lenstra–Pomerance: the 1984 algorithm was "the first factoring algorithm of which the expected running time was conjectured to be $L_n[\frac{1}{2}, 1 + o(1)]$, and it is now also the first algorithm for which that conjecture must be withdrawn".

An incorrect analysis of a subset-sum algorithm

Eurocrypt 2010 Howgrave-Graham–Joux claimed that it will "solve 1/2-unbalanced knapsacks in time $\widetilde{O}(2^{0.3113n})$".

An incorrect analysis of a subset-sum algorithm

Eurocrypt 2010 Howgrave-Graham–Joux claimed that it will "solve 1/2-unbalanced knapsacks in time $\widetilde{O}(2^{0.3113n})$".

2011 Becker–Coron–Joux: No, May and Meurer found a mistake. Correcting this mistake changes 0.3113 to 0.337. But here's a different algorithm obtaining 0.291, really!

An incorrect analysis of a subset-sum algorithm

Eurocrypt 2010 Howgrave-Graham–Joux claimed that it will "solve 1/2-unbalanced knapsacks in time $\widetilde{O}(2^{0.3113n})$".

2011 Becker–Coron–Joux: No, May and Meurer found a mistake. Correcting this mistake changes 0.3113 to 0.337. But here's a different algorithm obtaining 0.291, really!

2019 Esser–May: Here's another algorithm obtaining 0.255.

An incorrect analysis of a subset-sum algorithm

Eurocrypt 2010 Howgrave-Graham–Joux claimed that it will "solve 1/2-unbalanced knapsacks in time $\widetilde{O}(2^{0.3113n})$".

2011 Becker–Coron–Joux: No, May and Meurer found a mistake. Correcting this mistake changes 0.3113 to 0.337. But here's a different algorithm obtaining 0.291, really!

2019 Esser–May: Here's another algorithm obtaining 0.255.

Three months later: 2019 paper was withdrawn ("Issue with counting duplicate representations").

An incorrect analysis of an Ideal-SVP algorithm

Crypto 2019 Ducas–Plançon–Wesolowski: performance graph for an asymptotically useful quantum algorithm to attack Ideal-SVP; "reassuring" conclusion that "the cross-over point with BKZ-300 should not happen before ring rank $n \approx 6000$".

An incorrect analysis of an Ideal-SVP algorithm

Crypto 2019 Ducas–Plançon–Wesolowski: performance graph for an asymptotically useful quantum algorithm to attack Ideal-SVP; “reassuring” conclusion that “the cross-over point with BKZ-300 should not happen before ring rank $n \approx 6000$”.

2021: Online update radically revised the graph and changed “6000” to “2000”, crediting a six-person team for discovering a critical sign error inside the underlying attack analysis.

Do we care?

"The numbers don't matter unless they're feasible."

Do we care?

"The numbers don't matter unless they're feasible."

— Applications often choose bleeding-edge key sizes.
Small errors in exponents are dangerous.

Do we care?

"The numbers don't matter unless they're feasible."

— Applications often choose bleeding-edge key sizes.
Small errors in exponents are dangerous.

Furthermore, decisions of which cryptosystem to use are often based on very small differences in exponents: e.g., NTRU-509 costs less than Kyber-512, but NIST eliminated NTRU-509 (below AES-128?) and kept Kyber-512 (above AES-128?).

Do we care?

"The numbers don't matter unless they're feasible."

— Applications often choose bleeding-edge key sizes.
Small errors in exponents are dangerous.

Furthermore, decisions of which cryptosystem to use are often based on very small differences in exponents: e.g., NTRU-509 costs less than Kyber-512, but NIST eliminated NTRU-509 (below AES-128?) and kept Kyber-512 (above AES-128?).

Also, small differences can warp risk evaluation and resource allocation. e.g. Asiacrypt 2017 Chailloux–Naya-Plasencia–Schrottenloher incorrectly claimed quantum collision exponent $12n/25$, slightly below traditional non-quantum $n/2$.

Do we care? part 2

"Errors are rare."

Do we care? part 2

"Errors are rare."

— How rare? "Only 10% of attack analyses are wrong"?

Do we care? part 2

“Errors are rare.”

— How rare? “Only 10% of attack analyses are wrong”?
Aren’t we worried about the damage from that rate of errors?

Do we care? part 2

"Errors are rare."

— How rare? "Only 10% of attack analyses are wrong"?
Aren't we worried about the damage from that rate of errors?
And why exactly should we believe that the rate isn't higher?

Do we care? part 2

"Errors are rare."

— How rare? "Only 10% of attack analyses are wrong"? Aren't we worried about the damage from that rate of errors? And why exactly should we believe that the rate isn't higher?

Analogy from "provable security": 2007 Goldreich commented on "the unfortunate (**and rare**) cases in which flaws were found in published claimed 'proofs' (of security)" (boldface added). No quantification; no evidence.

Do we care? part 2

“Errors are rare.”

— How rare? “Only 10% of attack analyses are wrong”? Aren’t we worried about the damage from that rate of errors? And why exactly should we believe that the rate isn’t higher?

Analogy from “provable security”: 2007 Goldreich commented on “the unfortunate (**and rare**) cases in which flaws were found in published claimed ‘proofs’ (of security)” (boldface added). No quantification; no evidence. Many years later: Koblitz–Menezes surveyed many flawed “security proofs”.

Let's figure out how many proofs are wrong

Any *correct* proof can be explained to a computer.
Examples of systems to verify proofs: Coq (new name: Rocq), HOL4, HOL Light, Isabelle/HOL, Lean, Metamath, Mizar.

Let's figure out how many proofs are wrong

Any *correct* proof can be explained to a computer.
Examples of systems to verify proofs: Coq (new name: Rocq), HOL4, HOL Light, Isabelle/HOL, Lean, Metamath, Mizar.

If we take a *supposed* proof and go through this process, either we obtain high assurance that the proof is correct, or we find a problem with the proof.

Let's figure out how many proofs are wrong

Any *correct* proof can be explained to a computer.
Examples of systems to verify proofs: Coq (new name: Rocq), HOL4, HOL Light, Isabelle/HOL, Lean, Metamath, Mizar.

If we take a *supposed* proof and go through this process, either we obtain high assurance that the proof is correct, or we find a problem with the proof.

Does this sound expensive? It's less expensive than you think. The community can afford to do it for a random sample of proofs, giving clear evidence of the failure rate of proofs.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify the attack algorithm in the model of computation.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify the attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of the algorithm.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify the attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of the algorithm.
5. Fully specify the formula for the predicted success probability of the algorithm.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify the attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of the algorithm.
5. Fully specify the formula for the predicted success probability of the algorithm.
6. Fully specify the proof that the algorithm matches these predictions.

High assurance for a proof of attack effectiveness

Effectiveness $=$ (success probability, cost). Tasks:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify the attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of the algorithm.
5. Fully specify the formula for the predicted success probability of the algorithm.
6. Fully specify the proof that the algorithm matches these predictions.
7. Have a computer verify each step in the proof.

Wait, where's the proof?

The incorrect analyses from 1984 Schnorr–Lenstra, 2010 Howgrave-Graham–Joux, etc. never claimed to be proofs.

Wait, where's the proof?

The incorrect analyses from 1984 Schnorr–Lenstra, 2010 Howgrave-Graham–Joux, etc. never claimed to be proofs.

The analyses relied on estimates and heuristics and experiments. Some *fragments* of the analyses had proofs, but the (known) errors were outside these fragments.

Wait, where's the proof?

The incorrect analyses from 1984 Schnorr–Lenstra, 2010 Howgrave-Graham–Joux, etc. never claimed to be proofs.

The analyses relied on estimates and heuristics and experiments. Some *fragments* of the analyses had proofs, but the (known) errors were outside these fragments.

Clearly the plan of verifying proofs cannot give us high assurance for heuristic attack analyses, and cannot tell us the failure rate of those analyses.

Heuristic analyses are the normal situation

More examples of factorization algorithms:

- Trial division: proven.
- 1970 Shanks $n^{1/5+o(1)}$ algorithm: heuristic.
- 1974 Pollard $n^{1/4+o(1)}$ algorithm: proven but slower.
- 1974 Pollard $p - 1$ algorithm: heuristic.
- 1975 Pollard rho algorithm: heuristic.
- 1977 Schroeppel linear sieve: heuristic.
- 1981 Dixon random-squares method: proven but slower.
- 1982 Pomerance quadratic sieve: heuristic.
- 1987 Lenstra elliptic-curve method: heuristic.
- 1990 Pollard number-field sieve: heuristic.
- 1992 Lenstra–Pomerance method: proven but slower.

Ignoring heuristic speedups would be dangerous!

A post-quantum example: lattices

Some exponents for attacking *n*-dimensional SVP:

- 2011: $0.384n$, heuristic.
- 2013: $0.3778n$, heuristic.
- 2014: $0.3774n$, heuristic.
- 2015: $0.337n$, heuristic.
- 2015: $0.298n$, heuristic.
- 2015: $1.000n$, proven but slower.
- 2016: $0.292n$, heuristic.

A post-quantum example: lattices

Some exponents for attacking n-dimensional SVP:

- 2011: $0.384n$, heuristic.
- 2013: $0.3778n$, heuristic.
- 2014: $0.3774n$, heuristic.
- 2015: $0.337n$, heuristic.
- 2015: $0.298n$, heuristic.
- 2015: $1.000n$, proven but slower.
- 2016: $0.292n$, heuristic.

More heuristics appear in using SVP for BKZ,
and in other algorithms for attacking lattice problems,
especially for "structured lattices" arising from number fields.
e.g. STOC 2009 Gentry FHE system for power-of-2
cyclotomics is *conjectured* to be broken in quantum poly time.

This pattern is well known among experts

e.g. 1992 Lenstra: “The analysis of many algorithms related to algebraic number fields seriously challenges our theoretical understanding, and one is often forced to argue on the basis of heuristic assumptions that are formulated for the occasion. It is considered a relief when one runs into a standard conjecture such as the generalized Riemann hypothesis (as in [6, 15]) or Leopoldt’s conjecture on the nonvanishing of the p-adic regulator [60].”

High assurance for heuristic attack analyses

Even without proofs, can imagine the following process:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify each attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of each algorithm.
5. Fully specify the formula for the predicted success probability of each algorithm.

High assurance for heuristic attack analyses

Even without proofs, can imagine the following process:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify each attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of each algorithm.
5. Fully specify the formula for the predicted success probability of each algorithm.
6. Have a computer simulate each algorithm, comparing the observed cost to the prediction.

High assurance for heuristic attack analyses

Even without proofs, can imagine the following process:

1. Fully specify the model of computation and a cost metric.
2. Fully specify the problem under attack.
3. Fully specify each attack algorithm in the model of computation.
4. Fully specify the formula for the predicted cost of each algorithm.
5. Fully specify the formula for the predicted success probability of each algorithm.
6. Have a computer simulate each algorithm, comparing the observed cost to the prediction.
7. Have a computer simulate each algorithm, comparing the observed success probability to the prediction.

CryptAttackTester demonstrates feasibility

CryptAttackTester (CAT) includes formal specifications of

- a general-purpose model of computation and cost metric;
- examples of problems: (1) AES-128 key recovery, (2) the basic attack problem in code-based cryptography;
- case studies of attack algorithms in this model: (1) brute-force AES-128 key search, (2) information-set decoding (ISD), the state-of-the-art McEliece attack;
- formulas predicting the cost of each algorithm in this metric; and
- formulas predicting the success probability of each algorithm.

CAT includes a general-purpose simulator for this model of computation. Paper presents results for AES and ISD.

How do we check probability for an AES attack?

Generalize the problem to allow scaled-down experiments.

The problem has a parameter K, the number of key bits, used inside the code generating problem instances:

```
vector<bool> keybits;
for (bigint j = 0;j < K;++j)
  keybits.push_back(random_bool());

unsigned char keybytes[16];
for (bigint j = 0;j < 16;++j)
  keybytes[j] = 0;
for (bigint j = 0;j < 128 && j < K;++j)
  keybytes[j/8] += (int(keybits.at(j))<<int(j%8));
```

Attack algorithms are built from bit operations

CAT provides attack inputs as `vector<bit>`.
Attack builds a circuit from bit XOR, bit AND, etc.
For example, this attack subroutine does 8 bit operations:

```
typedef vector<bit> byte;

static byte byte_xor(byte c,byte d)
{
  byte result;
  for (bigint i = 0;i < 8;++i)
    result.push_back(c.at(i)^d.at(i));
  return result;
}
```

The attack interface

To add an attack to CAT:

- Add a function $\mathcal{A}$ that builds the attack circuit, given problem parameters and attack parameters. e.g. `aes128_enum`.

The attack interface

To add an attack to CAT:

- Add a function $\mathcal{A}$ that builds the attack circuit, given problem parameters and attack parameters. e.g. `aes128_enum`.
- Add a function $\mathcal{C}$ that predicts cost of the attack, given problem parameters and attack parameters. e.g. `aes128_enum_cost`.

The attack interface

To add an attack to CAT:

- Add a function $\mathcal{A}$ that builds the attack circuit, given problem parameters and attack parameters. e.g. `aes128_enum`.
- Add a function $\mathcal{C}$ that predicts cost of the attack, given problem parameters and attack parameters. e.g. `aes128_enum_cost`.
- Add a function $\mathcal{P}$ that predicts success probability of the attack, given problem parameters and attack parameters. e.g. `aes128_enum_prob`.

The attack interface

To add an attack to CAT:

- Add a function $\mathcal{A}$ that builds the attack circuit, given problem parameters and attack parameters. e.g. `aes128_enum`.
- Add a function $\mathcal{C}$ that predicts cost of the attack, given problem parameters and attack parameters. e.g. `aes128_enum_cost`.
- Add a function $\mathcal{P}$ that predicts success probability of the attack, given problem parameters and attack parameters. e.g. `aes128_enum_prob`.

CAT uses $\mathcal{A}$ to simulate the circuit on many inputs; compares observed effectiveness to the results of $\mathcal{C}$ and $\mathcal{P}$.

NIST's AES-128 estimates

In its 2016 call for post-quantum submissions, NIST specified AES-128 key search as a "floor" for security, and estimated "2^{143} classical gates" for an "optimal" AES-128 key-recovery attack. No details, and no definition of the set of "gates".

NIST's AES-128 estimates

In its 2016 call for post-quantum submissions, NIST specified AES-128 key search as a "floor" for security, and estimated "2^{143} classical gates" for an "optimal" AES-128 key-recovery attack. No details, and no definition of the set of "gates".

In its 2022 report, NIST wrote that, in "the *gate count* model", the "operations being counted are 'bit operations' that act on no more than 2 bits at a time and where each one-bit memory read or write is counted as one bit-operation" when "memory is read or written in a random access fashion".

Flaws in NIST's AES-128 estimates

Allowing a "one-bit memory read or write" for cost 1 allows many bit operations to be clumped into a single "gate": e.g., just 8 "gates" to compute an AES S-box. This easily reduces key-recovery cost to about 2^{140}.

Flaws in NIST's AES-128 estimates

Allowing a "one-bit memory read or write" for cost 1 allows many bit operations to be clumped into a single "gate": e.g., just 8 "gates" to compute an AES S-box. This easily reduces key-recovery cost to about 2^{140}.

CAT does not allow this type of cheating: CAT counts all bit operations involved in building a memory-access circuit. CAT shows that AES-128 key recovery with a simple brute-force attack takes under $2^{141.89}$ bit operations.

Flaws in NIST's AES-128 estimates

Allowing a "one-bit memory read or write" for cost 1 allows many bit operations to be clumped into a single "gate": e.g., just 8 "gates" to compute an AES S-box. This easily reduces key-recovery cost to about 2^{140}.

CAT does not allow this type of cheating: CAT counts all bit operations involved in building a memory-access circuit. CAT shows that AES-128 key recovery with a simple brute-force attack takes under $2^{141.89}$ bit operations.

These aren't very different from 2^{143}, but expect larger errors for more complicated attacks than brute-force key search.

Flaws in ISD estimates in the literature

ISD papers generally state costs using undefined concepts such as “work factor”, “elementary operations”, or “complexity”. This makes the cost claims formally meaningless.

But papers *sound like* they’re counting bit operations, often claiming different costs for the same algorithms, so clearly most of those claims are wrong.

Flaws in ISD estimates in the literature

ISD papers generally state costs using undefined concepts such as "work factor", "elementary operations", or "complexity". This makes the cost claims formally meaningless.

But papers *sound like* they're counting bit operations, often claiming different costs for the same algorithms, so clearly most of those claims are wrong.

e.g. For `mceliece348864`, one paper says $2^{149.91}$ for "BJMM"; another paper says 2^{142} for "BJMM" (below NIST's 2^{143}!). Different algorithms? Different attack-parameter optimizations? Different overestimates? Different underestimates? All of the above?

CAT's auditable ISD numbers

CAT has a completely defined `isd2` algorithm,
which covers BJMM and further optimizations.
Also a defined method of searching for attack parameters.

CAT's auditable ISD numbers

CAT has a completely defined `isd2` algorithm,
which covers BJMM and further optimizations.
Also a defined method of searching for attack parameters.

CAT's cost predictions and probability predictions cover various effects missed in the literature and are tested for small problem sizes against complete attack simulations.

CAT's auditable ISD numbers

CAT has a completely defined `isd2` algorithm,
which covers BJMM and further optimizations.
Also a defined method of searching for attack parameters.

CAT's cost predictions and probability predictions cover various effects missed in the literature and are tested for small problem sizes against complete attack simulations.

Predicts $2^{150.59}$ for `mceliece348864`.

CAT's auditable ISD numbers

CAT has a completely defined `isd2` algorithm,
which covers BJMM and further optimizations.
Also a defined method of searching for attack parameters.

CAT's cost predictions and probability predictions cover various effects missed in the literature and are tested for small problem sizes against complete attack simulations.

Predicts $2^{150.59}$ for `mceliece348864`.

For comparison: earlier paper's 2^{142} counted number of *input and output bits* for sorting, not the number of *bit operations*.