CryptAttackTester: formalizing attack analyses

Daniel J. Bernstein*
University of Illinois at Chicago
USA
Ruhr University Bochum
Germany

Tung Chou*
Academia Sinica
Taiwan

Abstract

Quantitative analyses of the costs of cryptographic attack algorithms play a central role in comparing cryptosystems, guiding the search for improved attacks, and deciding which cryptosystems to standardize. Unfortunately, these analyses often turn out to be wrong.

Formally verifying complete proofs of attack performance is a natural response but crashes into an insurmountable structural problem: there are large gaps between the best proven cost among known attack algorithms and the best conjectured cost among known attack algorithms. Ignoring conjectured speedups would be a security disaster.

This paper presents a case study demonstrating the feasibility and value of successfully formalizing what state-of-the-art attack analyses actually do. The input to this formalization is not a proof, and the output is not a formally verified proof; the formalization process nevertheless enforces clear definitions, systematically accounts for all algorithm steps, simplifies review, improves reproducibility, and reduces the risk of error.

Concretely, this paper's CryptAttackTester (CAT) software includes formal specifications of (1) a general-purpose model of computation and cost metric, (2) various attack algorithms, and (3) formulas predicting the cost and success probability of each algorithm. The software includes general-purpose simulators that systematically compare the predictions to the observed attack behavior in the same model. The paper gives various examples of errors in the literature that survived typical informal testing practices and that would have been immediately caught if CAT-enforced links had been in place.

The case study in CAT is information-set decoding (ISD), the top attack strategy against the McEliece cryptosystem. CAT formalizes analyses of many ISD algorithms, covering interactions between (1) high-level search strategies from Prange, Lee–Brickell, Leon, Stern, Dumer, May–Meurer–Thomae, and Becker–Joux–May–Meurer; (2) random walks from Omura, Canteaut–Chabaud, Canteaut–Sendrier, and Bernstein–Lange–Peters; and (3) speedups in core subroutines such as linear algebra and sorting.

1 Introduction

There is a long history of critical flaws in analyses of the performance of algorithms to attack cryptosystems. For example:

- The 1984 Schnorr–Lenstra factorization algorithm [109] was, in the words of 1992 Lenstra–Pomerance [79, page 484], "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".
- 2010 Howgrave-Graham–Joux [67] claimed "we can solve 1/2-unbalanced knapsacks in time $\tilde{O}(2^{0.3113n})$", and backed this up with a detailed algorithm analysis [67, Section 4]. However, in 2011, Becker–Coron–Joux [15, Section 2] reported that May and Meurer had found a mistake in [67], and that correcting this mistake changed 0.3113 to 0.337.
- 2017 Chailloux–Naya-Plasencia–Schrottenloher [44] stated that a generic quantum algorithm to find n-bit collisions had "a time-space product of $\tilde{O}(2^{12n/25})$", outperforming the well-known $n/2$ exponent for non-quantum parallel algorithms. However, Bernstein [21] pointed out that this $12n/25$ was a calculation error: the time-space product for the algorithm actually has exponent $13n/25$, above $n/2$.
- 2019 Esser–May [56] claimed subset-sum exponent 0.255, improving upon the best previous exponent (namely 0.291 from [15], improving upon the aforementioned 0.337). Three months later, a comment "Issue with counting duplicate representations" was added and the paper was withdrawn.
- 2019 Ducas–Plançon–Wesolowski [53, Figure 5] graphed performance of an asymptotically useful quantum algorithm to attack Ideal-SVP, and drew the "reassuring" conclusion that "the cross-over point with BKZ-300 should not happen before ring rank $n \approx 6000$". In 2021, an online update of [53] 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.

For [56], the error was caught at the preprint stage. For each of the other examples, the error was in a peer-reviewed paper in a high-profile publication venue. Many more examples are known.

The positive view is that each of these examples shows the scientific community successfully identifying and correcting an error. It is nevertheless concerning to see one example after another of an error playing a critical role in an announced attack analysis and not being caught until later, sometimes years later. Even more concerning is that today's processes for catching these errors are informal and haphazard; presumably the *known* error rate is an underestimate of the *actual* error rate. This procedural deficiency leaves real-world cryptography vulnerable to an important class of attacks; see Appendix A.

[1]Both authors contributed equally to this research. Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was funded by the Intel Crypto Frontiers Research Center; by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Excellence Strategy of the German Federal and State Governments—EXC 2092 CASA—390781972 "Cyber Security in the Age of Large-Scale Adversaries"; by the U.S. National Science Foundation under grant 1913167; by the Taiwan's Executive Yuan Data Safety and Talent Cultivation Project (AS-KPQ-109-DSTCP); and by the Taiwan's National Science and Technology Council (NSTC) grant 109-2222-E-001-001-MY3. Date of this document: 2023.06.14.

1.1. The obvious path to high assurance, and why the path fails for cryptanalysis. See [14] for a survey of exciting progress in formalization and automated verification of proofs, including security proofs for cryptographic protocols and correctness proofs for cryptographic software. It is natural to ask whether formally verified proofs can also address the deluge of errors in security analysis of the underlying mathematical primitives.

The obvious strategy to formally verify a proof of the effectiveness of any particular attack, where effectiveness is defined as the pair (success probability, cost), is as follows:

- Fully specify the model of computation and a cost metric.
- Fully specify the problem under attack.
- Fully specify the attack algorithm in the model of computation.
- Fully specify the formula for the predicted cost of the algorithm.
- Fully specify the formula for the predicted success probability of the algorithm.
- Fully specify the proof that the algorithm matches these predictions.
- Have a computer verify each step in the proof.

The first five steps are formally *stating* the claim of attack effectiveness. The sixth step takes the existing informal proof of the claim and turns it into a formal proof. The last step eliminates errors more reliably than humans do; this step requires all of the specifications to be in languages whose semantics are understood by the computer.

But what happens if the existing attack analysis isn't an informal proof, but merely a conjecture? The analyses mentioned above from [109], [67], [15], [56], and [53], for factorization and subset sum and Ideal-SVP, never claimed to be theorems: they are a different kind of science, relying on estimates and heuristics and experiments. Some *fragments* of the analyses had proofs (see, e.g., [67, Corollary 5]), but the errors were outside these fragments.

Similar comments apply to state-of-the-art attacks against many other cryptographic problems. The best proven effectiveness among known attacks is usually far worse than the best conjectured effectiveness among known attacks: see, e.g., [67] saying that 0.385 is the best exponent "for which we can prove the algorithm" but conjecturing that the algorithm reaches 0.3113. Focusing on proven effectiveness would, conjecturally, overestimate security levels—and would do so in a way that varies from one cryptosystem to another. Readers not familiar with these patterns should see Appendix B for many examples.

1.2. Formalizing and automating attack simulations. A proof is only one way to link a model of computation, a cost metric, an attack algorithm, a cost prediction, a problem, and a success-probability prediction. Another way is to simulate the attack in that model against that problem, comparing its observed success probability (over multiple simulation runs) to the prediction, and comparing its observed cost in the model to the prediction.

Formalizing and automating this process for a list of attacks would work as follows:

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

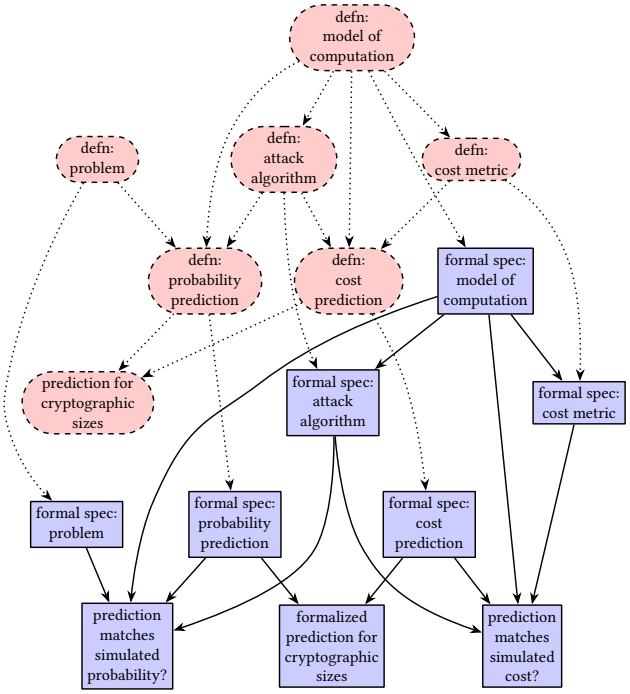

Figure 1: Data flow when an informal attack analysis (rounded dashed boxes) is supplemented with a formalization. Dotted edges are informal processes.

- Fully specify each attack algorithm in the model of computation.
- Fully specify the formula for the predicted cost of each algorithm.
- Fully specify the formula for the predicted success probability of each algorithm.
- Have a computer simulate each algorithm, comparing the observed cost to the prediction.
- Have a computer simulate each algorithm, comparing the observed success probability to the prediction.

Note that the model of computation, the cost metric, and the simulator can and should be attack-independent tools, for reusability and reviewability.

Attack simulations are already the central tool used in the literature to check conjectural attack analyses. *Formalizing* complete attack analyses would catch further errors, and would provide a clear structure for splitting reviews and risk analyses into simpler components. See Figure 1.

As examples of how attack simulations are already used, [67] tried its subset-sum algorithm, and [53] simulated its quantum Ideal-SVP algorithm. See Appendix C for how the aforementioned errors in [67] and [53] slipped past the experiments in those papers, but would have been stopped if the complete attack analyses had been formalized.

The literature sometimes presents what can be viewed as components of this type of formalization, at least for simple examples. Any software that computes predictions for cost and success probability

can be viewed as fully specifying formulas, modulo any relevant ambiguities in the programming language. The literature presents simulations checking some simple algorithms in clearly defined models of computation, and sometimes also checking cost formulas in clearly defined cost metrics. For example, [106] presents and checks a gate-level algorithm for reversible scalar multiplication, the main work inside an elliptic-curve version of Shor's algorithm.

However, this level of specification rapidly disappears as one moves to more complicated attack algorithms. It is not at all clear from the literature that it is feasible to formalize state-of-the-art cryptanalysis of unbroken cryptosystems.

1.3. The case of ISD. This paper focuses on one case study, called "information-set decoding" (ISD). This is the state-of-the-art attack strategy against a broad class of code-based cryptosystems, notably the well-known McEliece cryptosystem.

This paper's CryptAttackTester (CAT) software demonstrates feasibility, for this case study, of carrying out the entire linked series of formalization steps explained in Section 1.2. CAT includes formal specifications of (1) a general-purpose model of computation and cost metric, (2) the problem under attack, (3) a spectrum of ISD algorithms in this model, (4) formulas predicting the cost of each algorithm in this metric, and (5) formulas predicting the success probability of each algorithm. This paper carefully tunes the details of its ISD algorithms for performance in this metric.

CAT includes a general-purpose simulator for this model of computation. This paper reports what this simulator says about these cost and probability formulas for these ISD algorithms. In short, the cost predictions are perfect, and the success-probability predictions are close to perfect. This paper also quantifies the predicted effectiveness of these ISD algorithms for proposed McEliece parameters.

This simulator provides a clear framework for evaluating further ISD algorithms: add specifications of the algorithms and their cost/probability formulas, make sure the formulas are accurate in the simulations, and then use the formulas to compare to other algorithms. See Section 3. The framework is not limited to some preconceived notion of how ISD algorithms should be built: the same simulator can run arbitrary attack algorithms.

1.4. Reasons to take ISD as a case study. Among all proposed post-quantum public-key encryption systems, the McEliece cryptosystem has the strongest security track record. ISD was already known when the cryptosystem was introduced in 1978, and has always driven evaluations of the McEliece security level. Other known attack strategies have always been much slower than ISD, avoiding the worrisome situation of security being damaged by an improvement in any one of multiple competing lines of attack. Improvements in ISD since 1978 have made zero change in asymptotic McEliece exponents (for this asymptotic analysis see [31], [30, Section 1], and [119]), and have made only small changes in concrete exponents for security levels of interest (as Table 1 illustrates).

The fact that actual attack improvements are small is not a reason to expect analysis errors to be correspondingly small. For example, if an analysis misses an attack step, the magnitude of the error depends on how the cost of that step compares to the cost of other steps. If two different cost metrics are conflated, the magnitude of the error depends on the gap between the cost metrics. If there is a calculation error, the magnitude of the error can be arbitrarily large. These effects have no obvious connection to how stable attacks are.

When actual security levels are converging while errors are not, it becomes more and more likely for a *claimed* algorithm improvement to be the result of an error. This confuses readers regarding security risks, and warps the scientific process of searching for better attacks.

Appendix N gives examples of the magnitude of numerical variations among estimates of ISD attack costs, especially as a result of undocumented variations in which steps are counted and how costs are assigned to those steps. This paper's formalization systematically enforces counting all steps in a clearly defined cost metric, making it much easier to see *actual* algorithm improvements.

2 Choosing a model of computation and a cost metric

The literature contains many different definitions of the word "algorithm", and many different definitions of cost metrics for algorithms. Often two choices are polynomially equivalent in the sense that cost C under the first definition implies cost $C^{O(1)}$ under the second definition and vice versa, making the definitions interchangeable for, e.g., proving reductions among polynomial-time adversaries; but more precision is required when one wants to see, e.g., the difference between an attack taking time $2^{0.3n}$ and an attack taking time $2^{0.5n}$.

This section selects a particular model of computation and cost metric for this paper's case study. In short, the model is a conventional Boolean-circuit model, specifically with every $\leq$2-bit-to-1-bit function allowed as a cost-1 gate. Bit 0 and bit 1 and bit copying are free. The model and metric have a simple definition (see Section 2.1) and a straightforward formalization (see Section 3.6).

Conceptually, this paper's formalization process can start with other fully defined models of computation and cost metrics. There are many possibilities in the literature beyond this particularly easy Boolean-circuit model. Appendix E summarizes other common Boolean-circuit models. Appendix F considers validation of the particular model used here. Appendix G considers the more complicated possibility of using random-access-machine models (RAM models) rather than Boolean-circuit models.

2.1. The selected circuit model and cost metric. The following model of computation has two parameters: nonnegative integers A and B. The model expresses algorithms as circuits that map A bits of input to B bits of output.

An ***A*-bit-to-*B*-bit circuit** is a sequence $(C_A, C_{A+1}, \ldots, C_{A+L-1})$ such that (1) L is an integer with $L \geq B$ and (2) C_k, for each $k \in \{A, A+1, \ldots, A+L-1\}$, has the form $(\ell, F, i_0, \ldots, i_{\ell-1})$ where

- $\ell \in \{0, 1, 2\}$;
- F is a function from $\{0,1\}^\ell$ to $\{0,1\}$; and
- $i_0, \ldots, i_{\ell-1} \in \{0, 1, \ldots, k-1\}$.

The **cost** of the circuit is the number of k for which C_k has the form $(2, \ldots)$ or $(1, (x \mapsto 1-x), i)$.

The circuit is run as follows. The input bits, in order, are labeled $x_0, \ldots, x_{A-1}$. The circuit computes successively $x_A, \ldots, x_{A+L-1}$ by defining each x_k as $F(x_{i_0}, \ldots, x_{i_{\ell-1}})$ where $C_k = (\ell, F, i_0, \ldots, i_{\ell-1})$. The output consists of the bits $x_{A+L-B}, \ldots, x_{A+L-1}$ in that order.

3 Structure of the formalization

The output of this paper's process of formalizing ISD analyses is the CryptAttackTester (CAT) software package available from [24]. This section explains what CAT contains and how it relates to informal ISD analyses. See Appendix H for limitations in CAT.

3.1. External interface. The external interface of CAT consists of various functions that are automated: i.e., the software package includes software that computes these functions upon request. There are three classes of functions: *predictors*, *simulators*, and *explorers*.

The *predictors* are functions `predictedcost`, `predictedprob`, and `predictedcp`. Each function takes three inputs:

- A list of parameters for the problem under attack. For this ISD case study, a parameter list is a tuple (n, k, t), and the problem is to recover a secret weight-t vector $e \in \mathbb{F}_2^n$ given a matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ and given He. See Sections 3.4 and 3.5 for how this problem is formalized.
- An attack name. For this ISD case study, attacks are named by high-level search strategies (see Section 4): `isd0`, `isd1`, and `isd2`. There are also straightforward `bruteforce` and `bruteforce2` attacks as a baseline.
- A list of parameters for the attack: for example, the number of attack iterations. Each attack has its own list of parameter names.

The `predictedcp` function outputs the predicted cost and probability of that attack, with those attack parameters, against those problem parameters. The functions `predictedcost` and `predictedprob` output cost and probability separately, saving time in some applications of `predictedcp`.

The *simulators* are functions `circuitcost`, `circuitprob`, and `circuitexample`. These take the same inputs as `predicted*`, but also output the observed cost and success probability of the attack circuits for comparison to the predictions, or an input-output example for `circuitexample`. All of these simulators are internally built from a single unified simulator.

For example, if these functions are asked about `attack=isd0 L=0 P=0 I=1` for $(n, k, t) = (48, 36, 2)$, they report that the predicted circuit cost is 12325, the observed circuit cost is 12325, the predicted circuit success probability is slightly above 0.058, and the circuit was observed to succeed in 984 out of 16227 trials. The observed success probability in this example is almost 0.061; this is not a surprising deviation from the prediction for this number of trials.

If the observed success probability is outside $[0.9p, 1.1p]$, where p is the predicted success probability, then `circuitprob` also returns an alert. The number of trials, 16227 in this example, is automatically chosen by `circuitprob` as 1000 for $p > 1/2$ or $\lceil 1000(1-p)/p \rceil$ for $p \leq 1/2$, so alerts are rare when predictions are accurate. Increasing the 1000 ("`trialfactor`") inside `circuitprob` carries out more trials; this has the disadvantage of more run time but the advantage of being able to detect smaller-scale inaccuracies in the predictions.

The *explorers* formalize various procedures for exploring the space of circuits:

- `problemparams` generates a sequence of problem parameters to use for tests (not necessarily the full space of problem parameters). For ISD, this sequence includes examples of cryptographic interest such as $(3488, 2720, 64)$ and various scaled-down examples such as $(48, 36, 2)$.
- `attackparams`, given problem parameters, returns a list of pairs, where each pair consists of an attack name and an attack parameter list applicable to those problem parameters.
- `searchparams` uses heuristics to search for improved attack parameters, as measured by the ratio between predicted cost and predicted probability.

Note that, as in the literature, there is no guarantee that optimal attack parameters have been found (except when parameter spaces are very small). Perhaps there are large tradeoffs between the performance of an attack and the time spent finding the attack, as in the examples in [27]. The point of `searchparams` is to clearly specify a typical search process, not to claim that this process is optimal.

3.2. Comparison to previous ISD analyses. Estimates in the previous ISD literature come from "estimators" having the same basic data flow as the predictors in CAT: see, e.g., [97], [13], [55], and [57]. These estimators convert problem parameters, attack names, and attack parameters (found by search processes with varying levels of documentation) into predictions of cost and success probability.

One advantage of the predictors in CAT is that there are complete definitions of algorithms all the way down through the model of computation and cost metric, giving clear semantics to the predictions. Another advantage is that the details of the analyses of cost and probability account for various effects that the literature does not account for. Various highlights of these analyses are explained later in this paper, and the full analyses are formalized inside the predictors in CAT.

Both of these advantages follow from the central structural advantage of CAT: namely, CAT also includes a simulator, with the simulator results compared to the predictions. The simulator enforces clarity in definitions of cost metrics, and clarity in the definitions of the attacks under consideration. The comparisons provide reasons to believe that the attack analyses used the same clear cost metrics, accounted for all attack steps, and accounted for all major probability factors.

As noted in Section 1, these are not proofs. Perhaps an algorithm-analysis inaccuracy is (1) invisible in small simulations but (2) much larger at cryptographic sizes. However, this type of formalization would have rapidly caught all of the peer-reviewed errors listed in Section 1; see Appendix C for details of two examples. For the case of ISD (see Appendix N), a large underestimate in [13] would have been rapidly caught by small simulations, and smaller underestimates that this paper points out in [55] and [57] were caught by earlier stages in this paper's formalization process.

The ISD literature includes software for some slices of the space of ISD algorithms. The critical advantage of this paper's software is that it measures every attack in the same clearly defined cost metric used for cost predictions, allowing direct comparisons between measurements and predictions. For comparison, [29] includes predictions and software, but notes that "optimizing CPU cycles is different from, and more difficult than, optimizing the simplified notion of 'bit operations' " used in the predictions; [29] does not attempt to close the gap.

This paper's software also includes many more ISD algorithms than previous software. For example, Stern's algorithm [114], a specific example of `isd1`, was improved using random walks in [42] and more advanced random walks in [29]; but the literature has not analyzed the impact of random walks upon subsequent search strategies such as `isd2`. In this paper, every high-level ISD search strategy is systematically combined with the full space of random-walk strategies. As another example, this paper's collision searches systematically support tradeoffs between buffer sizes, collision-checking effort, and collision-finding probability; see the `QU`, `PE`, and `WI` parameters in Section 5.

3.3. The process of adding more attacks. Algorithm designers go beyond looking at the effectiveness of existing attacks: they consider the details of how attacks work, and search for more effective attacks. The internal structure of CAT is designed to assist in inspection of attack details and in adding further attacks to the same framework.

For example, the attack named `isd2` is defined by a function named `isd2`. This is accompanied by an `isd2_cost` function that predicts the attack cost, an `isd2_prob` function that predicts the attack probability, and an `isd2_params_valid` function that defines which parameter lists are valid for this attack. There is also an `isd2_params` function that generates a sequence of parameter lists for this attack; the first parameter list is the starting point for `searchparams`, and all parameter lists are output by `attackparams`.

Adding another attack and its analysis to the same framework means writing a function that constructs the circuit for the attack, along with functions for cost predictions etc., and adding these new functions to the list of attacks in CAT.

Attacks do not have to be built from scratch. Often an attack can be built as a simple modification of another attack. Many components of the attacks and analyses inside CAT are already provided as general-purpose subroutines, such as `sorting` to build a sorting circuit, `sorting_cost` to return the cost of that circuit, and `collision_average` to return a heuristic estimate for the number of collisions found by sorting two lists under a limit on the collision distance; this limit is the `WI` parameter in Section 5.11.

3.4. Formalizing the problem. A problem is formalized in CAT as a function `psgen` that, given a parameter list, returns a pair (P, s), where each of P and s is a bit vector of length determined by the parameter list. A problem is also accompanied by functions `params` (used inside `problemparams`) and `num{inputs,outputs}` (which are used as explained in Appendix K).

A trial in `circuitprob`, for any particular problem and any particular attack circuit, uses `psgen` to generate a pair (P, s), and asks whether input P to the circuit produces output s. Informally, the problem is to find the "secret" information s given the "public" information P.

This class of problems includes, for example, the one-wayness ("OW-CPA") problem for any public-key-encryption system (PKE) equipped with bit-vector encodings. In the OW-CPA problem, a public key is generated as specified by the PKE; a plaintext is chosen uniformly at random from the plaintext space; a ciphertext is obtained by encrypting the plaintext under that public key; the attacker's goal is to recover the plaintext, given the public key and the ciphertext. This fits straightforwardly into CAT, with the public bit vector P encoding the public key and the ciphertext, and the secret bit vector s encoding the plaintext.

For interesting problems, `psgen` is randomized, so there are many choices of (P, s). Currently randomness comes from a DRBG with known seeds for reproducibility. The DRBG is the `mt19937_64` DRBG built into the C++ programming language, not a DRBG designed to be cryptographically strong. The DRBG is isolated inside a small `random` module in CAT so that it can be easily replaced. Preliminary experiments with other DRBGs have not detected any influence of the DRBG choice upon any of the attacks in CAT. It would also be possible to run simulations using a hardware RNG.

3.5. An example of a problem. The `uniformmatrix` problem in CAT has integer parameters (n, k, t) where $n \geq 8$; $0.7n \leq k \leq 0.8n$; and $k = n - t\lceil \log_2 n \rceil$. The secret information is an n-bit vector $e \in \mathbb{F}_2^n$ generated uniformly at random subject to the constraint of having Hamming weight t. The public information is a uniform random matrix $T \in \mathbb{F}_2^{(n-k)\times k}$ and a ciphertext He, where $H \in \mathbb{F}_2^{(n-k)\times n}$ is defined as an identity matrix followed by T.

This problem has the virtue of simplicity, and the virtue of matching what is considered in typical ISD analyses. This problem is identical to the OW-CPA problem for a PKE where key generation returns public key T and an empty secret key; encryption of plaintext e produces ciphertext He; and decryption always fails. Decryption does not appear in the OW-CPA definition and is not used in CAT.

For small parameters, public keys in this PKE are easily distinguishable from public keys in the McEliece PKE (which uses a key-generation procedure different from this PKE and a decryption procedure different from this PKE), and in particular plaintexts often collide under encryption in this PKE while they never do in the McEliece PKE. Appendix K analyzes how these collisions reduce attack success probabilities. For large parameters, all known methods of distinguishing the two PKEs are much slower than known OW-CPA attacks; see generally [8].

The choice of putting an identity matrix before T matches [8]. Internally, the attacks in CAT rearrange input columns (at cost 0) to put the identity matrix after T, and rearrange output bits accordingly (also at cost 0). This makes some attack steps slightly easier to describe.

Structurally, CAT is not limited to the `uniformmatrix` problem. See Appendix I for another problem already included in CAT, an AES-128 key-search problem.

3.6. Formalizing the model of computation. Informally, each attack is an algorithm that, given problem parameters and attack parameters, constructs a circuit as in Section 2.1. The circuit is applied to a problem instance to produce an output. The algorithm that generates a circuit is called a "meta-circuit" in the following paragraphs, to avoid confusion with the algorithm expressed by the circuit per se.

The meta-circuit is formalized as a function in C++. The function arguments are problem parameters, attack parameters, and a problem instance. The problem instance consists of a bit vector (P above) of length determined by the parameters. Each bit uses a `bit` class defined centrally by `bit.h`. The function returns an output,

namely a vector of `bit` values, where again the vector length is determined by the parameters.

To construct a circuit, the meta-circuit simply carries out operations on `bit` values; `bit.h` automatically tracks the circuit cost. For example, one of the lowest-level subroutines is a `half_adder` subroutine that adds two bits `a` and `b` to obtain a two-bit sum, namely a bottom bit `s` and a carry bit `c`. The code for the subroutine says `s = a ^ b` and `c = a & b`.

The usual C/C++ notation is supported for AND (`&`), OR (`|`), XOR (`^`), and NOT (`~`); the `half_adder` example used this notation for XOR and AND. All 2-bit operations are supported with conventional gate names such as `a.andn(b)`. Constructing `bit(0)` and `bit(1)` is free, as is copying.

(Section 2.1 also allows uninteresting cost-1 2-input gates that output 0, 1, or a copy of an input. For completeness of the formalization of the model, note that these gates can be computed as `bit(0)&bit(0)`, `bit(1)&bit(1)`, or `b&b` respectively.)

As another low-level example, there is a `bit_vector_ixor` function that XORs a vector `w` into a vector `v`:

```
static inline void bit_vector_ixor(vector<bit> &v,
                                   const vector<bit> &w)
{
        assert(v.size() == w.size());
        for (bigint i = 0; i < v.size(); i++)
                v.at(i) = v.at(i) ^ w.at(i);
}
```

Here `.at(i)` is bounds-checked C++ array access, which is used systematically throughout CAT to avoid the well-known risk of accidents from non-bounds-checked `[i]`. (Protection against accidents should not be confused with protection against malice: adding malicious attack code to CAT can exploit the known DRBG seeds, overwrite results, destroy files, etc.)

CAT provides an abstract integer type, `bigint`, to shield formalizations from the limited-size "integral" types in C++ (and from the resulting ambiguities: the size limits are compiler-dependent). Internally, CAT implements `bigint` via GMP, and GMP's overhead creates a considerable `circuit*` slowdown,[2] presumably increasing the risk that prediction errors will be missed.[3] It is not easy to compare this risk to the risk of error arising from using, e.g., a 64-bit integral type for inner loops. It is well known that languages can in principle make `bigint` much faster, with multiple code paths and range analysis to automatically replace `bigint` with a fast fixed-width type in most cases, but so far this has received less compiler support than analogous hoisting of bounds checks.

Meta-circuits do not inspect the values of the bits that they are operating upon, so the circuits that they build are independent of the inputs, as in the informal description of an attack. The probability simulator `circuitprob` automatically runs circuits on many inputs at once in bitsliced form.

[2] An experiment that modified CAT to instead use `long long` for vector indices in meta-circuits reduced the time for CAT's `isdsims.py` script (see Section 6) by an order of magnitude and produced the same output.

[3] Faster simulations allow a larger limit on the simulation size that the user can afford for any given amount of CPU time spent on simulations. Perhaps this larger limit makes a prediction error visible. See generally Appendix L.

4 ISD variants

This section and Section 5 describe the attacks covered in CAT against the problem defined in Section 3.5. This section emphasizes the central mathematical objects computed in these ISD variants. Section 5 emphasizes the construction and optimization of circuits for subroutines to compute those objects.

See Section 6 for examples of choosing ISD variants to attack specific problem sizes. These choices depend on costs and success probabilities for the complete ISD circuits, including the layers in this section and in Section 5. A closer look at the details shows interactions across layers: for example, understanding the cost of linear-algebra circuits is important for seeing the benefit of the new random-walk parameter Y introduced below. These choices are also influenced by the model of computation and cost metric: for example, the literature already indicates that accounting for two-dimensional or three-dimensional communication costs tends to favor fewer levels of collision search and a smaller p parameter.

4.1. Relationship to the literature. Before presenting the attacks, this section summarizes how these attacks relate to previous work.

After Prange's original ISD algorithm in 1962 [102], ISD variants developed in two major directions. One direction is improvements in linear-algebra costs; this includes random walks through information sets (credited in [46] to Omura), combinatorial searches to reuse linear algebra for many tests (Lee–Brickell [75]), and testing only a limited number of bits (Leon [80]).

Omura's random walks changed one position in an information set to obtain a new information set. Canteaut–Chabaud [41] and Canteaut–Sendrier [42] considered an analogous modification of Stern's algorithm, and used Markov chains to analyze the impact. Bernstein–Lange–Peters [29] showed that changing multiple positions at a time further improves Stern's algorithm, at the expense of a more complicated Markov-chain analysis.

The other major direction is asymptotically better combinatorial searches, including 1 level of collision search (as in Stern [114] and Dumer [54]), 2 levels of collision search (as in May–Meurer–Thomae [82], which adapted Howgrave-Graham–Joux [67] to decoding), and allowing collisions with partial cancellations (as in Becker–Joux–May–Meurer [18], which adapted Becker–Coron–Joux [15] to decoding).

The attacks and analyses in CAT systematically integrate random walks with 0, 1, or 2 levels of collision search. The attack description below first explains the random walks, and then explains the three search options. For 2 levels, collisions with partial cancellations are supported in CAT, and the analysis of the "$C = 1$" option described below appears to be new. Some search techniques that the literature describes as small improvements are not included in CAT: ball-collision decoding as in [30], 3 levels of collision search as in [18], and nearest-neighbor search as in [83].

The random walks in CAT are more general than the random walks in [29]. The X parameter in this paper is the number of positions changed, matching the "c" parameter in [29]; the new Y parameter in this paper reduces the number of positions considered for a change. Taking the maximum possible choice of Y matches the random walks in [29]. Taking much smaller Y creates a noticeable chance of a failed information-set update spoiling all

subsequent iterations, but periodic **resets** in this paper limit the impact of failures: a completely new information set is chosen every RE iterations, starting from the original input matrix, producing a new **chain** of information sets. Manual parameter selection would take RE large enough to hide the occasional reset costs compared to per-iteration search costs, and would take Y somewhat above $X + \log_2 \mathsf{RE}$ so that it is rare for a chain to fail.

4.2. Notation. By $I \oplus J$, we denote the symmetric difference of two sets I and J. Given a nonnegative integer d, we denote by $[d]$ the set $\{0, 1, \ldots, d\}$. Vectors, if not stated otherwise, are considered as column vectors over $\mathbb{F}_2$. By $v \mid\mid v'$, we denote the result of concatenating vectors v and v'. By $\mathrm{wt}(v)$ we denote the Hamming weight of a vector v. By v_i we denote entry i (the index starts from 0) of a vector v. We denote by u_i the ith unit vector, of which the length depends on the context. We denote by $\mathrm{vec}(I, d)$ the vector v in $\mathbb{F}_2^d$ such that $v_i = 1$ if and only if $i \in I$. By $\mathrm{nrows}(A)$ we denote the number of rows of a matrix A. By $\mathrm{ncols}(A)$ we denote the number of columns of a matrix A. By A_i we denote row i of a matrix A. By $A[i]$ we denote column i of a matrix A. Similarly, by $A[I]$, where I is a set of integers, we denote $\sum_{i \in I} A[i]$. Given an integer d, a matrix A with at least d columns, and a vector s of length $\mathrm{nrows}(A)$, we denote by $\mathcal{S}_d(A, s)$ the set

$$\left\{ \left(s + A[I], I\right) \mid I \subseteq \left[\mathrm{ncols}(A) - 1\right], |I| = d \right\}.$$

Similarly, we denote by $\mathcal{S}_d(A, A', s, s')$ the set

$$\left\{ \left(s + A[I], s' + A'[I], I\right) \mid I \subseteq \left[\mathrm{ncols}(A) - 1\right], |I| = d \right\}.$$

4.3. Attack overview. Each attack takes two inputs $H \in \mathbb{F}_2^{(n-k)\times n}$ and $s \in \mathbb{F}_2^{n-k}$ and outputs a vector $e \in \mathbb{F}_2^n$, where the last $n - k$ columns of H form an identity matrix (i.e., H is in **systematic form**). Each attack tries to ensure that e is a vector of Hamming weight t satisfying $He = s$.

(Note that the problem in Section 3.4 is different. The identity matrix there is at a different position, and success in the OW-CPA problem requires recovering a particular preimage of s under H, which is a narrower notion than finding an arbitrary preimage when there are multiple preimages. See Appendix K.)

Each attack consists of a sequence of **iterations** followed by a simple **post-processing phase**. The number of iterations is specified by a parameter $\mathsf{IT} > 0$. Each iteration consists of two phases: a **column-permutation phase** and a **search phase**. The column-permutation, search, and post-processing phases are described below.

Each attack has a parameter $\mathsf{FW} \in \{0, 1\}$. If $\mathsf{FW} = 1$ then the attack begins by extending H to include a row $(1, 1, \ldots, 1)$, extending s to include a corresponding bit t mod 2, reducing the new H to systematic form (and failing if this reduction fails), adjusting s accordingly, and reducing k to $k - 1$. For literature using the known sum of elements of e to reduce k by 1, see [48, page 57, "zero mean"] for lattices, [49, full version, Section 6.3] for lattices, and [57, Section 3.1] for codes.

4.4. Column-permutation phase. Each column-permutation phase applies in-place operations to a matrix $\tilde{H}$ and a vector $\tilde{s}$, where $\tilde{H}$ and $\tilde{s}$ are scrambled versions of H and s, respectively. The operations applied to $\tilde{s}$ are simply the same as the row operations that are applied to $\tilde{H}$, so below we only talk about operations that are applied to $\tilde{H}$. The column-permutation phase uses four attack parameters: $\ell \geq 0$, $\mathsf{RE} > 0$, $X > 0$, and $Y \geq X$.

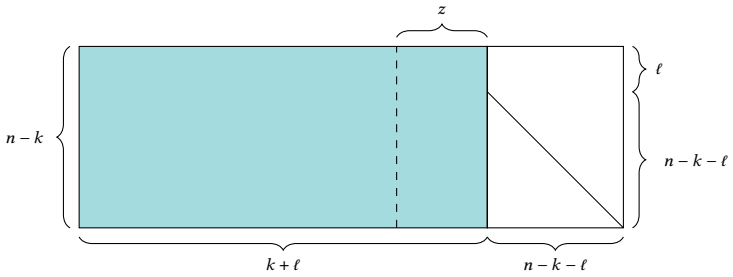

Figure 2: A $(n-k) \times n$ matrix in generalized systematic form.

By definition, a matrix A is in **generalized systematic form** if the last $\mathrm{nrows}(A) - \ell$ columns of the matrix consist of ℓ zero rows and an identity matrix. Figure 2 depicts an $(n-k) \times n$ matrix in generalized systematic form. (This form is also used in previous papers such as [82], usually without a name.) Each column-permutation phase aims to permute the columns of $\tilde{H}$ in a sufficiently random way while keeping $\tilde{H}$ in generalized systematic form.

The operations carried out in a column-permutation phase depend on the iteration number, as described below.

4.4.1. First iteration. If the iteration number is 0, the column-permutation phase first sets two variables $\tilde{H}$ and $\tilde{s}$ to H and s, respectively. Then, for each $i \in \{0, \ldots, \ell - 1\}$ in order, $b_{i,j}\tilde{H}_i$ is added to $\tilde{H}_j$ for each $j \neq i$, where each $b_{i,j} \in \mathbb{F}_2$ is chosen randomly. (Without the row additions, almost all entries in $\tilde{H}[k], \ldots, \tilde{H}[k + \ell - 1]$ would be 0, which in experiments produces considerable deviations from the predicted success probability.)

4.4.2. Starting a subsequent chain. If the iteration number is a non-zero multiple of RE, the column-permutation phase sets $\tilde{H}$ and $\tilde{s}$ to H and s, respectively. Then, the column-permutation phase permutes the columns of $\tilde{H}$ randomly, reduces $\tilde{H}$ to row-echelon form, and permutes the columns of $\tilde{H}$ to bring it to systematic form. Finally, each of the first ℓ rows is added to other rows in a random way, as in the first iteration.

4.4.3. Inside a chain. If the iteration number is not a multiple of RE, the column-permutation phase consists of three steps. These steps are designed to save bit operations by permuting only a small set of columns of $\tilde{H}$ instead of all columns. See Figure 3.

The first step applies a random permutation to the first $k + \ell$ columns of $\tilde{H}$. It then applies another random permutation to the last $n - k - \ell$ columns of $\tilde{H}$, and the same permutation to the last $n - k - \ell$ rows of $\tilde{H}$.

The second step applies row operations to rows $\tilde{H}_\ell, \ldots, \tilde{H}_{\ell+X-1}$ so that the $X \times Y$ submatrix formed by the first Y columns of the resulting rows is in reduced row-echelon form. It then permutes the columns $\tilde{H}[0], \ldots, \tilde{H}[Y-1]$ and $\tilde{H}[k+\ell], \ldots, \tilde{H}[k+\ell+X-1]$ so that the intersection between $\tilde{H}_\ell, \ldots, \tilde{H}_{\ell+X-1}$ and $\tilde{H}[k + \ell], \ldots, \tilde{H}[k +$

Figure 3: The attack parameters X and Y.

$\ell + X - 1]$ becomes an identity matrix. It then uses row operations to bring $\tilde{H}$ to generalized systematic form.

The third step works in the same way as the first step, making new choices of random permutations.

Ensuring that all X columns are exchanged with new columns is the "type 3" approach described in [29, "Analysis of the number of iterations"]. Considering only Y choices of new columns allows a smaller column-permutation circuit.

4.5. Search and post-processing phases. After each column-permutation phase, there is a permutation matrix P and an invertible matrix A such that

$$\tilde{H} = AHP,\ \tilde{s} = As.$$

Consequently, given P and any weight-t vector $\tilde{e}$ that satifies $\tilde{H}\tilde{e} = \tilde{s}$, it is easy to compute a weight-t vector $e = P\tilde{e}$ such that $He = HP\tilde{e} = s$. The goal of each search phase is to find such $\tilde{e}$ given $\tilde{H}$ and $\tilde{s}$, while the goal of the post-processing phase is to derive $e = P\tilde{e}$.

The matrix P is represented as a vector $\pi = (\pi_0, \ldots, \pi_{n-1}) \in \mathbb{Z}^n$ where $P[i] = u_{\pi_i}$. Whenever $\tilde{H}$ is set to H (inside the first iteration of each chain), π is set to $(0, 1, \ldots, n-1)$; whenever a column permutation is applied to $\tilde{H}$, the same permutation is applied to entries of π. Whenever a solution for $\tilde{e}$ is found in a search phase, π is stored into a **solution buffer**, along with some data from which $\tilde{e}$ can be derived. The post-processing phase derives $\tilde{e}$ from the data in the solution buffer and computes e as $P\tilde{e}$.

Three options for the search phase are described below: `isd0`, `isd1`, and `isd2`. The reader may wish to interpret each "$S \cdots \subseteq \cdots$" below as "$S \cdots = \cdots$" for an initial understanding of the attacks, but optimizing the new `QU`, `PE`, and `WI` parameters in Section 5 usually produces smaller subsets.

4.6. Failure to maintain generalized systematic form. Updates to the solution buffer are controlled by an update-permitted bit. If the $X \times Y$ matrix from which the column-permutation phase computes reduced row-echelon form is not full rank then the column-permutation phase fails to bring $\tilde{H}$ to generalized systematic form. The column-permutation phase then clears the update-permitted bit. Each new chain sets the update-permitted bit to 1.

4.7. `isd0`: 0 levels of collision search. The following text describes the search phase in CAT's `isd0` attack. There are three attack parameters that matter for the search, called p, ℓ, and z. This attack includes, for example, Prange's original ISD algorithm (parameters $p = 0$ and $\ell = 0$), the Lee–Brickell algorithm ($p > 0$ and $\ell = 0$), and Leon's algorithm ($\ell > 0$).

The first $k + \ell - z$ columns of $\tilde{H}$ are viewed as

$$\begin{pmatrix} T^{(0)} \\ T^{(1)} \end{pmatrix} \in \mathbb{F}_2^{(n-k)\times(k+\ell-z)},$$

where $\operatorname{nrows}(T^{(0)}) = \ell$ and $\operatorname{nrows}(T^{(1)}) = n - k - \ell$. Similarly, $\tilde{s}$ is considered as $\tilde{s}^{(0)} \mid\mid \tilde{s}^{(1)}$, where $\tilde{s}^{(0)} \in \mathbb{F}_2^{\ell}$ and $\tilde{s}^{(1)} \in \mathbb{F}_2^{n-k-\ell}$.

In the case $\ell > 0$, each search phase first computes

$$S^{(1)} \subseteq \left\{ I \mid (0, I) \in \mathcal{S}_p(T^{(0)}, \tilde{s}^{(0)}) \right\}.$$

Then, for each $I \in S^{(1)}$, the search phase computes $v = \tilde{s}^{(1)} - T^{(1)}[I]$ and checks if $\operatorname{wt}(v) = t - p$.

In the case $\ell = 0$, for each (v, I) in $\mathcal{S}_p(T, \tilde{s})$, the search phase checks if $\operatorname{wt}(v) = t - p$. Here T is the first $k - z$ columns of $\tilde{H}$.

Either way, if the check passes, then $\tilde{H}\tilde{e} = \tilde{s}$ must hold for the weight-t vector

$$\tilde{e} = (\operatorname{vec}(I, k + \ell - z) \mid\mid (0, \ldots, 0) \mid\mid v) \in \mathbb{F}_2^n.$$

The search phase then stores I and v in the solution buffer, so that the post-processing phase can derive $\tilde{e}$.

4.8. `isd1`: 1 level of collision search. CAT's `isd1` attack again has three attack parameters that matter for the search, called p', ℓ, and z. This attack includes, e.g., Stern's algorithm (with $z = \ell$) and Dumer's algorithm (with $z = 0$). The parameters p' and ℓ are required to be positive. The parameter p' in `isd1` is analogous to p in `isd0` in how it controls list sizes, but `isd1` uses these lists to search for $2p'$ errors while `isd0` uses these lists to search for p errors.

The search phase in `isd1` works as follows. Matrices $T^{(0)}, T^{(1)}$ and vectors $\tilde{s}^{(0)}, \tilde{s}^{(1)}$ are defined in the same way as in `isd0`, and we consider

$$T^{(i)} = \begin{pmatrix} T_L^{(i)} & T_R^{(i)} \end{pmatrix},$$

where $\texttt{ncols}(T_L^{(i)}) = \lfloor (k + \ell - z)/2 \rfloor$ and $\texttt{ncols}(T_R^{(i)}) = \lceil (k + \ell - z)/2 \rceil$.

Each search phase first computes two sets

$$S_L = \mathcal{S}_{p'}(T_L^{(0)}, 0),\ S_R = \mathcal{S}_{p'}(T_R^{(0)}, \tilde{s}^{(0)}).$$

Then a collision search between S_L and S_R is carried out to find

$$S^{(1)} \subseteq \left\{ (I_L, I_R) \mid (v, I_L) \in S_L \text{ and } (v, I_R) \in S_R \text{ for some } v \right\}.$$

For each $(I_L, I_R) \in S^{(1)}$, the search phase computes $w = \tilde{s}^{(1)} - (T_L^{(1)}[I_L] + T_R^{(1)}[I_R])$ and checks if $\operatorname{wt}(w) = t - 2p'$. If so, $\tilde{H}\tilde{e} = \tilde{s}$ must hold for the weight-t vector

$$\tilde{e} = (\operatorname{vec}(I_L, \lfloor k+\ell-z)/2 \rfloor) \mid\mid \operatorname{vec}(I_R, \lceil k+\ell-z)/2 \rceil) \mid\mid (0, \ldots, 0) \mid\mid w)$$

in $\mathbb{F}_2^n$. The search phase then stores I_L, I_R, w in the solution buffer, so that the post-processing phase can derive $\tilde{e}$.

4.9. `isd2`: 2 levels of collision search. Attack parameters in CAT's `isd2` attack include $\ell_0 > 0$ and $\ell_1 > 0$, with ℓ defined as $\ell_0 + \ell_1$; z; $p'' > 0$; $p' \in \{0, 2, \ldots, 2p''\}$; $C \in \{0, 1\}$; and $D \in \{1, \ldots, 2^{\ell_0}\}$.

The case $C = 0$ with $p' = 2p''$ is due to 2011 May–Meurer–Thomae [82] (MMT). The case $C = 0$ with $p' < 2p''$ is due to 2012 Becker–Joux–May–Meurer [18] (BJMM). The case $C = 1$ ignores p' and is essentially [63, Table 3], but the analysis in [63] treats this algorithm as succeeding only when the MMT algorithm does,

whereas the CAT analysis accounts for further success cases in the algorithm.

The first $k+\ell-z$ columns of $\tilde{H}$ are viewed as

$$\begin{pmatrix} T^{(0)} \\ T^{(1)} \\ T^{(2)} \end{pmatrix} \in \mathbb{F}_2^{(n-k)\times(k+\ell-z)},$$

where $\mathrm{nrows}(T^{(0)}) = \ell_0$, $\mathrm{nrows}(T^{(1)}) = \ell_1$, and $\mathrm{nrows}(T^{(2)}) = n-k-\ell$. Similarly, $\tilde{s}$ is considered as $\tilde{s}^{(0)} \,||\, \tilde{s}^{(1)} \,||\, \tilde{s}^{(2)}$, where $\tilde{s}^{(0)} \in \mathbb{F}_2^{\ell_0}$, $\tilde{s}^{(1)} \in \mathbb{F}_2^{\ell_1}$, $\tilde{s}^{(2)} \in \mathbb{F}_2^{n-k-\ell}$. We further consider

$$T^{(i)} = \begin{pmatrix} T_L^{(i)} & T_R^{(i)} \end{pmatrix},$$

where $\mathrm{ncols}(T_L^{(i)}) = \lfloor (k+\ell-z)/2 \rfloor$ and $\mathrm{ncols}(T_R^{(i)}) = \lceil (k+\ell-z)/2 \rceil$, respectively.

The attack parameter D is used to specify the size of a set $S_\Delta \subseteq \mathbb{F}_2^{\ell_0}$. For each $\Delta \in S_\Delta$, each search phase first computes

$$\begin{aligned} S_L^{(0)} &= \mathcal{S}_{p''}(T_L^{(0)}, T_L^{(1)}, 0, 0), \\ S_R^{(0)} &= \mathcal{S}_{p''}(T_R^{(0)}, T_R^{(1)}, \Delta, 0), \\ \hat{S}_R^{(0)} &= \mathcal{S}_{p''}(T_R^{(0)}, T_R^{(1)}, \tilde{s}^{(0)} + \Delta, \tilde{s}^{(1)}). \end{aligned}$$

Then, a collision search between $S_L^{(0)}$ and $S_R^{(0)}$ is performed to build

$$S^{(1)} \subseteq \left\{ (w_L + w_R, I_L, I_R) \mid (v, w_L, I_L) \in S_L^{(0)}, (v, w_R, I_R) \in S_R^{(0)} \right\}.$$

Similarly, a collision search between $S_L^{(0)}$ and $\hat{S}_R^{(0)}$ is performed to build

$$\hat{S}^{(1)} \subseteq \left\{ (\hat{w}_L + \hat{w}_R, \hat{I}_L, \hat{I}_R) \mid (v, \hat{w}_L, \hat{I}_L) \in S_L^{(0)}, (v, \hat{w}_R, \hat{I}_R) \in \hat{S}_R^{(0)} \right\}.$$

Once $S^{(1)}$ and $\hat{S}^{(1)}$ are obtained, another collision search is performed to build

$$S^{(2)} \subseteq \left\{ (I_L, \hat{I}_L, I_R, \hat{I}_R) \mid (v, I_L, I_R) \in S^{(1)}, (v, \hat{I}_L, \hat{I}_R) \in \hat{S}^{(1)} \right\}$$

if $C = 1$. If $C = 0$, $S^{(2)}$ is built in a similar way except that each $(I_L, \hat{I}_L, I_R, \hat{I}_R) \in S^{(2)}$ needs to satisfy two additional constraints $|I_L \oplus \hat{I}_L| = p'$ and $|I_R \oplus \hat{I}_R| = p'$.

Once $S^{(2)}$ is obtained, for each $(I_L, \hat{I}_L, I_R, \hat{I}_R) \in S^{(2)}$, the search phase then computes $w = \tilde{s}^{(2)} - \left(T_L^{(2)}[I_L \oplus \hat{I}_L] + T_R^{(2)}[I_R \oplus \hat{I}_R]\right)$ and checks

- in the case $C = 0$: whether $\mathrm{wt}(w) = t - 2p'$;
- in the case $C = 1$: whether $|I_L \oplus \hat{I}_L| + |I_R \oplus \hat{I}_R| + \mathrm{wt}(w) = t$.

If so, $\tilde{H}\tilde{e} = \tilde{s}$ holds for the weight-t vector

$$\tilde{e} = \left(\mathrm{vec}(I_L \oplus \hat{I}_L, \lfloor k+\ell-z \rfloor) \,||\, \mathrm{vec}(I_R \oplus \hat{I}_R, \lceil k+\ell-z \rceil) \,||\, (0, \ldots, 0) \,||\, w\right)$$

in $\mathbb{F}_2^n$. The search phase then stores $I_L, \hat{I}_L, I_R, \hat{I}_R, w$ in the solution buffer so that the post-processing phase can derive $\tilde{e}$.

5 Circuits for the ISD variants

This section explains how the circuits in CAT handle the subroutines needed for Section 4: reducing matrices to echelon form, finding collisions, etc.

There are many improvements here compared to naive circuit designs. Some of the improvements show how ISD subroutines can exploit known techniques such as fast sorting networks. Some of the improvements provide new tradeoffs between cost and success probability for ISD subroutines: for example, taking this section's `WI` parameter to be small accelerates collision-finding at the expense of sometimes missing collisions, a tradeoff that turns out to be worthwhile.

5.1. Queues. An important low-level tool is a fixed-length queue of fixed-size vectors. Conditionally pushing a vector into a queue, given a bit c, means pushing the vector into the queue if $c = 1$, and not doing anything if $c = 0$. To carry out this operation, first conditionally move each vector in the queue except for the last one to the next vector, and then conditionally move the new vector to the first vector. To conditionally move variable $b_0 \in \mathbb{F}_2$ to variable $b_1 \in \mathbb{F}_2$ given the condition bit $c \in \mathbb{F}_2$, simply overwrite b_1 with $b_1 + c(b_0 + b_1)$ ("MUX"), at cost 3. In other words, if the queue consists of q vectors each of d bits, conditionally pushing a new vector into the queue costs $3dq$.

5.2. Hamming-weight computation. Let $v \in \mathbb{F}_2^d$. Computing $\mathrm{wt}(v)$ costs 0 if $d = 1$ and costs 2 (`half_adder`) if $d = 2$. For $d \geq 3$, CAT uses the algorithm described in [36], which works as follows. Let α be the largest integer such that $\alpha + 1 \leq d$ and $\alpha + 1$ is power of 2. Let $\beta = d - 1 - \alpha$. The algorithm recursively computes w_α and w_β, the Hamming weights of the first α coordinates and the next β coordinates of v, respectively. Finally, $\mathrm{wt}(v)$ is computed as $w_\alpha + w_\beta + v_{d-1}$, in a way that v_{d-1} serves as the carry-in bit.

5.3. Computing $\mathcal{S}_d(A, v)$. CAT computes $\mathcal{S}_d(A, v)$ by computing the leaves of a tree. Each node in the tree is of the form $\mathcal{N}(v, A, I) := (v + A[I], I)$, where $I \subseteq [\mathrm{ncols}(A) - 1]$ and $|I| \leq d$. The root of the tree is defined as $\mathcal{N}(v, A, \emptyset)$. The children of the root are defined as

$$\mathcal{N}\big(v, A, \{d-1\}\big), \ldots, \mathcal{N}\big(v, A, \{\mathrm{ncols}(A) - 1\}\big).$$

The children of a node $\mathcal{N}(v, A, I)$ with $0 < |I| < d$ are defined as

$$\mathcal{N}\big(v, A, I \cup \{d - |I| - 1\}\big), \ldots, \mathcal{N}\big(v, A, I \cup \{\min(I) - 1\}\big).$$

A node $\mathcal{N}(v, A, I)$ is considered as a leaf node if $|I| = d$. The leaf nodes form $\mathcal{S}_d(A, v)$. CAT computes each non-root node from its parent using exactly 1 vector addition. In this way, under the condition that $d \leq 10$ and $100 \leq \mathrm{ncols}(A) \leq 10000$, on average it takes no more than 1.11 vector additions to compute each element in $\mathcal{S}_d(A, v)$.

5.4. Random-access memory (RAM) operations. Sometimes CAT needs to compute $A[i]$ given a matrix A and an index i, where i depends on the entries in H and s. Similarly, sometimes CAT need to set $A[i]$ to v given a matrix A, an index i, and a vector v. The circuits for these **RAM read** and **RAM write** operations imitate real-world RAM circuits. For example, each RAM read operation is carried out by computing the root of a binary tree, where the columns of A form the leaves. Each non-leaf node with left child x and right child y is of value $(1-b)x + by$, where b is a specific bit in i: b is the least significant bit of i if the node is in the second lowest level of the tree, and b is the second least significant bit if the node is in the third lowest level of the tree, and so on. The value of each node with only 1 child is same as that of the child.

5.5. Sorting. Knuth's "merge exchange" algorithm [70] sorts a list L of d elements using $\Theta(d \log^2 d)$ compare-and-exchange operations, with small hidden constants, and is used for sorting in CAT. (Other algorithms are known using $\Theta(d \log d)$ compare-and-exchange operations, but with much larger Θ constants.) Each compare-and-exchange opration exchanges $L[i]$ and $L[j]$ if $L[i] > L[j]$, for some i, j such that $i < j$.

Sorting is used for various operations, in particular for collision search. See Section 5.11 for more details on collision search.

5.6. Computing the sum of specific columns. Often CAT needs to compute $A[I]$ for a matrix A and a set of indices I. It uses one of two approaches to compute $A[I]$. The first one is to simply obtain the corresponding columns by carrying out $|I|$ RAM reads and then compute the sum of the columns. The second one is to first compute $\mathrm{vec}(I, \mathrm{ncols}(A))$ via sorting and then multiply by A. For any given input size, CAT automatically predicts the cost of both approaches and selects the lower-cost approach.

5.7. Computing the size of symmetric difference. One of the operations required in `isd2` is computation of $|I \oplus J|$, where each of I and J is a set of p'' indices. Here CAT first sorts the indices in I and J together to form a sorted list L of length $2p''$ and sets a variable v to $(1, \ldots, 1) \in \mathbb{F}_2^{2p''}$. Then, for $i = 0, \ldots, 2p'' - 2$, if $v_i = v_{i+1} = 1$ and $L[i] = L[i+1]$, v_i and v_{i+1} are both set to 0. Finally, $|I \oplus J|$ is obtained as $\mathrm{wt}(v)$.

5.8. Computing reduced row-echelon form. To compute reduced row-echelon form of a matrix $A \in \mathbb{F}_2^{a \times b}$ where $a \le b$, CAT uses the following algorithm from [45]:

(1) Set $r = 0$.
(2) Set v to the logical OR of $A_r, \ldots, A_{a-1}$.
(3) Find the index j of the first nonzero entry in v. If $v = 0$, set j to any value in $\{0, \ldots, b-1\}$.
(4) For $i \in \{r+1, \ldots a-1\}$, $A_r \leftarrow A_r + A_i \cdot (1 - A_{r,j})$.
(5) For $i \in \{0, \ldots, a-1\} \setminus \{r\}$, $A_i \leftarrow A_i + A_r \cdot A_{i,j}$.
(6) If $r + 1 < a$, increase r by 1 and go back to Step 2.

Unrolling eliminates r, so Steps 1 and 6 cost 0. Step 2 is carried out using $(a - r - 1)b$ ORs. Step 4 and 5 are carried out using RAM operations: since j is known only after Step 3, RAM operations are used to read $A_{r,j}$ from A_r and read $A_{i,j}$ from A_i. A minor optimization mentioned in [45], and used in CAT, is to make Steps 2, 3, 4, 5 only work on entries in $A[r], \ldots, A[b-1]$.

In Step 3, it is required to find the index j of the first 1 in a vector of length b. If $b = 1$, j is simply set to 0. Now assume $b \ge 2$. Let α be the integer such that $2^\alpha < b$ and $2^{\alpha+1} \ge b$. Note that j can be represented as a vector of $\alpha + 1$ bits. To compute j, CAT carries out the following steps for $i = \alpha, \ldots, 0$:

(1) Compute j_i as $(1 - v_0) \cdots (1 - v_{2^i - 1})$.
(2) For all d such that $d < 2^i$ and $d + 2^i < b$, conditionally move v_{d+2^i} to v_d by considering j_i as the condition bit.

Note that if $v = 0$, we might have $j \ge b$ after the 3 steps are carried out. The circuits still compute reduced row-echelon form correctly in this case because RAM reads ensure that an entry of A_r or A_i will be obtained even when $j \ge b$.

5.9. Permuting columns. In each column-permutation phase, it is required to permute some columns of $\tilde{H}$ in a random way. Each circuit permutes the columns in a deterministic way, which is chosen randomly from all possible ways to permute the columns when the circuit is generated. This is simply copying data, at cost 0. (The wiring used here would be visible in a cost metric that accounts for communication costs.)

When the iteration number is a nonzero multiple of `RE`, the random column permutation is followed by a reduction to row-echelon form, and then by a conversion to systematic form, which works as follows. Let the column indices of the pivots be $i_0, \ldots, i_{n-k-1}$, where $i_0 < i_1 < \cdots < i_{n-k-1}$. To bring $\tilde{H}$ to systematic form, simply swap $\tilde{H}[i_{n-k-1}]$ with $\tilde{H}[n-1]$, swap $\tilde{H}[i_{n-k-2}]$ with $\tilde{H}[n-2]$, and so on, using $n-k$ RAM reads and $n-k$ RAM writes. Similarly, when the iteration number is not a multiple of `RE`, CAT carries out X RAM reads and X RAM writes to permute $\tilde{H}[0], \ldots, \tilde{H}[Y-1]$ and $\tilde{H}[k+\ell], \ldots, \tilde{H}[k+\ell+X-1]$.

5.10. Search phase in `isd0`. Denote by $\mathcal{E}(c)$ a bit which is of value 1 if and only if the statement c holds. To compute $S^{(1)}$ in `isd0`, for each $(v, I) \in \mathcal{S}_p(T^{(0)}, \tilde{s}^{(0)})$, CAT computes $\mathcal{E}(v = 0)$ and conditionally pushes I into a queue of size `QU`, where `QU` is an attack parameter. Every time `PE` elements in $\mathcal{S}_p(T^{(0)}, \tilde{s}^{(0)})$ are checked, where `PE` is another attack parameter such that `PE` $\ge$ `QU`, for each I in the queue, CAT computes $v = \tilde{s}^{(1)} - T^{(1)}[I], \mathcal{E}(\mathrm{wt}(v) = t - p)$ and conditionally stores (I, v) into the solution buffer. After all the elements in the queue are checked, the queue is cleared so that the next `PE` elements in $\mathcal{S}_p(T^{(0)}, \tilde{s}^{(0)})$ can be processed.

Note that every I such that $(0, I) \in \mathcal{S}_p(T^{(0)}, \tilde{s}^{(0)})$ will be pushed into the queue, but it might be kicked out from the queue, which is why $S^{(1)}$ might not be equal to the corresponding superset. The attack parameters `QU` and `PE` allow these circuits to trade efficiency (in terms of cost) for success probability, and vice versa. See Appendix J.1 for how CAT predicts queue-loss probabilities.

5.11. Search phase in `isd1`. In each search phase of `isd1`, to find collisions between S_L and S_R, CAT first sorts the elements in $\{(v, I, 0) \mid (v, I) \in S_L\}$ and $\{(v, I, 1) \mid (v, I) \in S_R\}$ together, using the following ordering: $(v, I, b) > (v', I', b')$ means that (1) $v > v'$ in lexicographic order or (2) $v = v'$ and $b' > b$. Let the sorted list be L, and, for an attack parameter `WI` > 0, define

$$S_{L,R} = \left\{(L[i], L[i+d]) \mid d \in \{1, \ldots, \texttt{WI}\}\right\}.$$

For each element $((v, I, b), (v', I', b')) \in S_{L,R}$ in a random order, CAT computes $\mathcal{E}(v = v', b \ne b')$ and conditionally pushes (I, I') into a queue of size `QU`. Every time `PE` elements in $S_{L,R}$ are checked, for each (I, I') in the queue, CAT computes $w = s^{(1)} - (T_L^{(1)}[I] + T_R^{(1)}[I'])$, $\mathcal{E}(\mathrm{wt}(w) = t - 2p)$ and conditionally stores (I, I', w) into the solution buffer. After all the elements in the queue are checked, the queue is cleared so that the next `PE` elements in $S_{L,R}$ can be processed.

The use of queues is as in `isd0`; the `WI` parameter provides another tradeoff between probability and cost. See Appendix J.2 for how CAT predicts window-loss probabilities.

5.12. Search phase in `isd2`. In each search phase in `isd2`, $S_L^{(0)}$, $S_R^{(0)}$, and $\hat{S}_R^{(0)}$ are generated for each $\Delta \in S_\Delta \subseteq \mathbb{F}_2^{\ell_0}$. As $S_L^{(0)}$ is independent of Δ, it is generated at the beginning of the search phase and simply reused for all Δ's. $S_R^{(0)}$ and $\hat{S}_R^{(0)}$, unlike $S_L^{(0)}$, can change for different Δ's. To save bit operations for generating $S_R^{(0)}$ and $\hat{S}_R^{(0)}$, CAT uses a Gray code to ensure that any two consecutive Δ's differ in only 1 bit. In this way, whenever a new Δ is used, it takes only 1 NOT to update each element in $S_R^{(0)}$ and $\hat{S}_R^{(0)}$.

The computations of $S^{(2)}$ from $S^{(1)}$ and $\hat{S}^{(1)}$, of $S^{(1)}$ from $S_L^{(0)}$ and $S_R^{(0)}$, and of $\hat{S}^{(1)}$ from $S_L^{(0)}$ and $\hat{S}_R^{(0)}$ in `isd2` work the same way as the computation of $S^{(1)}$ from S_L and S_R in `isd1`, using separate `WI1`, `QU1`, `PE1`, `WI2`, `QU2`, `PE2` parameters for the two levels.

6 Numerical results

This section presents and compares various numbers produced by this paper's formalization. Section 6.1 compares observed effectiveness of the simulated circuits for various small n to predicted effectiveness, finding a very close match. Section 6.2 quantifies what the predictions say at cryptographic sizes, specifically regarding (1) the security levels of the Classic McEliece parameter lists from [8] and (2) comparisons among ISD algorithms.

Appendix L considers ways that inaccuracies could have appeared in the predictions while avoiding detection by the simulations in Section 6.1.

6.1. Comparing simulations to predictions. The CAT package includes an `isdsims.py` script to run the experiments described below, and an `isdsims-graph.py` script to convert the results of the experiments into the graph shown in Figure 4.

The experiments are as follows:

- For each $n \in \{16, 18, 20, \ldots, 128\}$, and for each integer $t \geq 1$ such that $k = n - t\lceil \lg n \rceil$ satisfies $0.7 \leq k/n \leq 0.8$, use `searchparams` to heuristically search for parameters for `isd0` with $p = 0$ and $\ell = 0$, `isd0` with $\ell = 0$, `isd0` without restrictions, `isd1`, `isd2` with $C = 0$, and `isd2` with $C = 1$, in each case with `FW` chosen as 1 and with `(IT,RE)` chosen in three different ways: $(1, 1)$ or $(2, 1)$ or $(4, 4)$. This produces a sequence of attack parameter lists.
- For each attack parameter list, use `circuitcost` to compare the observed cost of the simulated circuit to the predicted cost. This raises an alert if the costs are not identical. (No cost alerts appeared.)
- Also, for each attack parameter list, use `circuitprob` with `trialfactor = 100000` and `probfactor = 100` to run many experiments with the simulated circuit and compare the observed success probability to the predicted success probability. Setting `probfactor = 100` skips circuits with success probability below 1%; concretely, this means skipping some of the larger `isd0` circuits.

The results of the probability comparison are shown in the graph in Figure 4. Each circuit produces one dot in the graph, where the horizontal position of the dot is n, the shape of the dot indicates t, and the vertical position of the dot is the ratio between circuit cost and observed success probability. An arrow coming from the left of the dot shows the ratio between circuit cost and predicted success probability. An arrow coming from the right of the dot shows the ratio between circuit cost and a simplified prediction of success probability, where the simplification omits a correction for collisions; see Appendix K. The two predictions differ by at most $(\binom{n-k}{t} - 1)/2^{n-k}$, which converges rapidly to 0 as n and t increase.

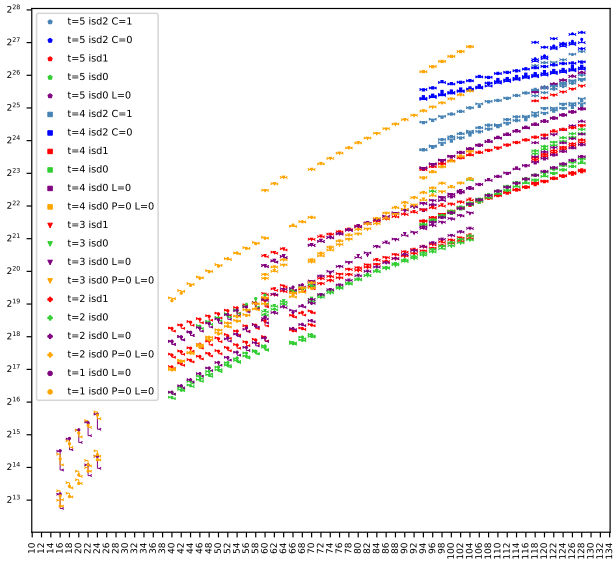

Figure 4: Accuracy of predictions of success probability for various attack circuits for small n. Horizontal axis: n. Vertical axis: circuit cost divided by success probability. Each dot shows experimentally observed successes from the simulated circuit with `trialfactor = 100000`. The arrow from the left of the dot shows the probability prediction. The arrow from the right of the dot shows a simplified probability prediction without a collision correction.

For small n with $t = 1$, the graph shows the prediction understating circuit effectiveness by about 0.1 bits (while the simplified prediction overstates circuit effectiveness), with a maximum error below 0.2 bits. The prediction generally becomes more accurate as n and t increase within the range of the graph, although the arrival of `isd2` for $t = 4$ is again accompanied by measurable deviations. Note that if predicted success probabilities are accurate then, with `trialfactor = 100000`, the ratio between observed success probability and actual success probability will have standard deviation considerably below 1%.

As expected, the graph (of observed values and of predicted values) shows large jumps upwards as t increases from 1 to 2 to 3 to 4 to 5, and gentler increases with n. For each problem parameter with $t \leq 3$, the smallest cost/probability ratios in the graph are from `isd0`; `isd1` begins to take over at $t = 4$. There are a few cases where the graph shows `searchparams` finding slightly better results when n is increased; presumably more comprehensive parameter searches would move more dots slightly downwards.

6.2. Predictions for cryptographic sizes. The CAT package includes scripts `isdpredict1.py` and `isdpredict2.py` to search for attack parameters for cryptographic sizes, along with a script `isdpredict-table.py` to turn the results into Table 1 below.

The Classic McEliece documentation selects five parameter lists (n, k, t), namely $(3488, 2720, 64)$; $(4608, 3360, 96)$; $(6688, 5024, 128)$; $(6960, 5413, 119)$; $(8192, 6528, 128)$. CAT takes 47 hours on a dual EPYC 7742 (with Core Performance Boost disabled) to collect data for all of these. The results for 6688 and 6960 are almost identical; to save space, the table shows just the smaller size 6688. For comparison, the script also covers $(n, k, t) = (1284, 1020, 24)$, the size of the largest challenge broken in [57] using `isd2`, and $(n, k, t) = (1347, 1047, 25)$, the size of a larger challenge broken in 2023 using `isd1` (see [23]); the table includes 1284.

For each parameter list, the table reports, for various circuits, the predicted ratio between circuit cost and success probability. Table entries in columns `isd` through C indicate attack constraints.

Table 1 is split into several sections. The first section, just one row, is Prange's original ISD algorithm: `isd0` with `RE` $= 1$, $\ell = 0$, and $p = 0$. The next section is for general `isd0`, including the Lee–Brickell use of p and Leon's use of ℓ. The next section is `isd1`, including collision searches from Stern and Dumer. All of this is still from the 1980s, except that random walks at the time were limited to Omura's $X = 1$ for `isd0`. Random walks do not matter for large p, p', p'' but provide an interesting speedup for small p, p', p''; see the table rows with and without the `RE` $= 1$ restriction.

There are, finally, three sections for `isd2`. The first is for $p' = 2p''$ with $C = 0$, as in the MMT paper, although this paper does better using random walks and many speedups in subroutines. The second is for $p' = 2p'' - 2$ with $C = 0$, an example of the BJMM paper with two levels, although again this paper includes more improvements. The third is for $C = 1$, where the CAT analysis appears to be the first analysis in the literature; note that the $C = 1$ attack ignores p', so taking $p' < 2p''$ would make no difference here.

Each section beyond the first is split into lines with different values of p or p' or p'', the parameter controlling the starting list sizes, so that the reader can see the drop in circuit costs as this parameter increases to an optimal value. The minimal cost in each section is highlighted in blue.

The asymptotics in the literature already indicate that the optimum for this parameter gradually increases with n in the McEliece context (see, e.g., [31, Sections 4–5]). CAT provides precise, fully defined data on this point for concrete sizes.

The full attack parameters for each table row are defined as follows. Starting from the parameter restrictions shown in the table plus `FW` $= 1$, the `isdpredict1.py` script runs `searchparams` for `IT` $= 1$ to find an initial list of parameters. For each $r \in \{0, 1, \ldots, 24\}$, the `isdpredict2.py` script then runs `searchparams` again to search for (X, Y) subject to `RE` $= 2^r$, `IT` $= 2^{r+16}$ (this ensures that the reset costs will be counted within a 2^{-16} error), and all other parameters matching the output from `isdpredict1.py`. The lowest cost/probability ratio across r is used for the table, except for the `RE` $= 1$ rows, which use $r = 0$.

Compared to the original Prange algorithm (first row), the smallest exponent shown in the table is 15% lower for $n = 3488$, reflecting the overall impact of many years of algorithmic improvements. It is remarkable that most of this change was already achieved in

Table 1: Logarithm base 2, rounded to 2 digits, of the ratio between predicted cost and predicted success probability for various attack circuits. See text for details.

isd	RE	ℓ	p	p'	p''	C	1284	3488	4608	6688	8192
0	1	0	0				85.99	177.26	221.14	299.99	338.10
0		0	0				78.62	168.34	211.56	290.00	328.15
0	1	0	1				79.39	169.34	213.02	291.29	329.02
0		0	1				75.24	163.70	206.54	284.50	322.49
0	1	0	2				78.13	166.70	209.62	287.59	325.62
0		0	2				78.07	166.63	209.51	287.50	325.56
0		0	3				81.53	170.14	213.13	291.11	329.17
0	1		1				80.47	170.53	214.24	292.50	330.21
0			1				74.46	163.06	205.87	283.94	321.87
0	1		2				76.27	165.12	208.51	286.32	323.72
0			2				72.54	159.93	202.30	279.88	317.66
0			3				75.89	163.31	205.68	283.34	321.17
1	1			1			76.83	165.65	209.02	286.83	324.27
1				1			71.11	158.39	201.17	278.63	316.07
1	1			2			71.66	157.33	199.60	276.39	313.45
1				2			71.48	156.96	198.93	275.71	312.97
1				3			73.16	157.43	199.27	275.53	312.47
1				4			75.00	157.95	199.69	275.41	312.03
1				5			77.19	158.64	200.24	275.41	311.69
1				6			80.73	159.48	200.91	275.55	311.52
1				7			87.30	160.50	201.74	275.82	311.44
1				8			93.88	161.68	202.71	276.23	311.50
1				9			100.26	163.03	203.83	276.74	311.69
2	1			2	1	0	72.37	158.93	201.45	278.24	315.21
2				2	1	0	70.95	156.26	198.61	275.13	312.22
2	1			4	2	0	72.71	155.41	197.29	278.84	309.65
2				4	2	0	72.67	155.38	197.20	278.74	309.52
2				6	3	0	75.84	156.27	202.70	274.09	308.43
2				8	4	0	80.31	157.92	198.04	272.84	307.72
2				10	5	0	86.58	158.19	199.07	272.32	307.04
2				12	6	0	98.04	160.02	200.07	271.62	306.18
2				14	7	0		161.95	201.72	271.53	305.11
2				16	8	0		165.39	203.25	272.00	304.62
2				18	9	0		167.83	206.08	272.97	306.20
2	1			2	2	0	75.07	163.29	203.50	283.28	317.05
2				2	2	0	75.00	163.13	203.18	282.95	316.84
2				4	3	0	72.74	156.54	203.14	275.02	310.93
2				6	4	0	73.88	156.14	197.40	271.98	308.41
2				8	5	0	77.13	154.76	196.19	270.81	306.29
2				10	6	0	84.28	155.15	195.91	268.91	304.23
2				12	7	0		155.75	196.43	267.77	302.11
2				14	8	0		157.89	196.92	267.30	300.68
2				16	9	0		159.07	198.11	267.41	301.39
2				18	10	0		163.40	201.31	268.05	301.06
2				20	11	0		167.53	202.41	269.99	300.49
2				22	12	0		171.35	205.45	270.17	301.19
2	1			2	1	1	72.59	158.59	201.70	278.31	315.21
2				2	1	1	70.90	156.26	198.21	275.14	312.21
2	1			4	2	1	70.99	158.62	199.22	278.45	309.19
2				4	2	1	70.95	158.46	198.90	278.12	309.06
2				6	3	1	71.07	154.21	200.67	272.72	307.34
2				8	4	1	72.45	154.17	195.37	270.42	305.78
2				10	5	1	75.35	152.45	193.88	267.79	303.99
2				12	6	1	82.39	151.78	192.78	266.34	301.82
2				14	7	1		150.84	191.95	264.40	299.18
2				16	8	1		150.91	191.56	263.57	296.93
2				18	9	1		150.59	190.62	260.44	296.45
2				20	10	1		151.46	191.41	261.13	294.64
2				22	11	1		151.77	190.50	260.40	292.46
2				24	12	1		152.91	191.18	259.02	291.14
2				26	13	1		154.04	190.55	259.44	290.83
2				28	14	1		156.08	192.21	258.54	289.99
2				30	15	1		159.08	193.16	258.69	289.31
2				32	16	1		165.51	194.12	258.29	287.21
2				34	17	1			195.99	257.36	288.00
2				36	18	1			197.89	258.34	287.50

the 1980s, most importantly from simply reducing linear-algebra costs; see Section 4.1's list of specific improvements and credits. Concretely, moving within `isd0` from Prange's algorithm to Leon's algorithm (with $p = 2$) gives 17 bits of improvement; moving from `isd0` to `isd1` (with $p' = 2$) gives just 3 more bits of improvement; and moving from `isd1` to `isd2` (with a much larger p'') gives just 6 bits of further improvement.

The numbers in this table are predictions of clearly defined mathematical objects: the model of computation, the cost metric, and the circuits are fully defined. This paper's formalization tests predictions directly against complete circuit simulations (see Figures 1 and 4), and the predictions account for various algorithm features that were missing from previous analyses. Readers are, however, cautioned to keep in mind that there are still risks of mispredictions, including risks arising from inadequate searches for circuits, risks arising from the structural limits of small-scale simulation as a form of verification, and risks arising from inaccuracies in the underlying model. See Appendix L.

The largest known issue is the following. Increasing p, p', p'' increases list size exponentially (e.g., for the `isd2` table entry with $C = 1$ and $p'' = 9$ for $n = 3488$, the first list has almost 2^{76} entries), and correspondingly increases the hardware mass and long-distance communication costs involved in collision searches inside `isd1` and `isd2`. This paper's formalization measures bit operations, including the bit operations involved in memory access, but does not account for hardware mass or communication costs. Cost metrics that account for those costs would favor lower-memory attacks.

References

[1] Report of the workshop on estimation of significant advances in computer technology, 1976. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nbsir76-1189.pdf.

[2] Sunny Cove: Intel's lost generation, 2022. URL: https://chipsandcheese.com/2022/06/07/sunny-cove-intels-lost-generation/.

[3] Scott Aaronson. Why isn't it more mysterious?, 2015. URL: https://web.archive.org/web/20150423085814/http://ideas.aeon.co/viewpoints/1829.

[4] Carlisle M. Adams and Henk Meijer. Security-related comments regarding McEliece's public-key cryptosystem. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 224–228, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany. doi:10.1007/3-540-48184-2_20.

[5] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 733–742, Portland, OR, USA, June 14–17, 2015. ACM Press. doi:10.1145/2746539.2746606.

[6] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[7] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the third round of the NIST Post-Quantum Cryptography Standardization Process, 2022. URL: https://csrc.nist.gov/publications/detail/nistir/8413/final.

[8] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

[9] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 583–613, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-64834-3_20.

[10] Ant Miner Store. Antminer S17 – 56TH/s, 2022. URL: https://web.archive.org/web/20220613183343/https://www.ant-miner.store/product/antminer-s17-56th/.

[11] Jean-Philippe Aumasson. Too much crypto. Cryptology ePrint Archive, Report 2019/1492, 2019. https://eprint.iacr.org/2019/1492.

[12] Eric Bach. Toward a theory of Pollard's rho method. *Information and Computation*, 90(2):139–155, 1991. doi:10.1016/0890-5401(91)90001-I.

[13] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. A finite regime analysis of information set decoding algorithms. *Algorithms*, 12(10):209, 2019. doi:10.3390/a12100209.

[14] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. In *2021 IEEE Symposium on Security and Privacy*, pages 777–795, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. doi:10.1109/SP40001.2021.00008.

[15] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-20465-4_21.

[16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM. doi:10.1137/1.9781611974331.ch2.

[17] Anja Becker, Nicolas Gama, and Antoine Joux. Solving shortest and closest vector problems: The decomposition approach. Cryptology ePrint Archive, Report 2013/685, 2013. https://eprint.iacr.org/2013/685.

[18] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-29011-4_31.

[19] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[20] Robert L. Benedetto, Dragos Ghioca, Benjamin Hutz, Pär Kurlberg, Thomas Scanlon, and Thomas J. Tucker. Periods of rational maps modulo primes. *Mathematische Annalen*, 355(2):637–660, 2013. doi:10.1007/s00208-012-0799-8.

[21] Daniel J. Bernstein. Quantum algorithms to find collisions, 2017. URL: https://blog.cr.yp.to/20171017-collisions.html.

[22] Daniel J. Bernstein. Cryptographic competitions, 2021. URL: https://cr.yp.to/papers.html#competitions.

[23] Daniel J. Bernstein. Solving the length-1347 McEliece challenge, 2023. URL: https://isd.mceliece.org/1347.html.

[24] Daniel J. Bernstein and Tung Chou. CryptAttackTester, 2023. https://cat.cr.yp.to.

[25] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[26] Daniel J. Bernstein, Nadia Heninger, Paul Lou, and Luke Valenta. Post-quantum RSA. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 311–329, Utrecht, The Netherlands, June 26–28, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-59879-6_18.

[27] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 321–340, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-42045-0_17.

[28] Daniel J. Bernstein and Tanja Lange. Two grumpy giants and a baby. In *ANTS X. Proceedings of the tenth algorithmic number theory symposium, San Diego, CA, USA, July 9–13, 2012*, pages 87–111. Berkeley, CA: Mathematical Sciences Publishers (MSP), 2013.

[29] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In Johannes Buchmann and Jintai Ding, editors, *Post-quantum cryptography, second international workshop, PQCRYPTO 2008*, pages 31–46, Cincinnati, Ohio, United States, October 17–19, 2008. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-88403-3_3.

[30] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-22792-9_42.

[31] Daniel J. Bernstein, Tanja Lange, Christiane Peters, and Henk C.A. van Tilborg. Explicit bounds for generic decoding algorithms for code-based cryptography. In *International Workshop on Coding and Cryptography (WCC 2009, Ullensvang, Norway, May 10–15, 2009)*, pages 168–180. Selmer Center, University of Bergen, 2009.

[32] Andrey Bogdanov, Donghoon Chang, Mohona Ghosh, and Somitra Kumar Sanadhya. Bicliques with minimal data and time complexity for AES. In Jooyoung Lee and Jongsung Kim, editors, *ICISC 14: 17th International Conference on Information Security and Cryptology*, volume 8949 of *Lecture Notes in Computer Science*, pages 160–174, Seoul, Korea, December 3–5, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-15943-0_10.

[33] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-25385-0_19.

[34] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. Improved classical and quantum algorithms for subset-sum. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 633–666, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-64834-3_22.

[35] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, April 2013. doi:10.1007/s00145-012-9124-7.

[36] Joan Boyar and René Peralta. The exact multiplicative complexity of the Hamming weight function. *Electronic Colloquium on Computational Complexity*, TR05-049, 2005. URL: https://eccc.weizmann.ac.il/eccc-reports/2005/TR05-049/index.html, arXiv:TR05-049.

[37] Richard P. Brent and H. T. Kung. The area-time complexity of binary multiplication. *J. ACM*, 28(3):521–534, 1981. doi:10.1145/322261.322269.

[38] Renée C. Bryce, Sreedevi Sampath, Jan B. Pedersen, and Schuyler Manchester. Test suite prioritization by cost-based combinatorial interaction coverage. *Int. J. Syst. Assur. Eng. Manag.*, 2(2):126–134, 2011. doi:10.1007/s13198-011-0067-4.

[39] James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

[40] Danielle Cadet. How the FBI invaded Martin Luther King Jr.'s privacy – and tried to blackmail him into suicide, 2014. URL: https://www.huffpost.com/entry/martin-luther-king-fbi_n_4631112.

[41] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[42] Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology – ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 187–199, Beijing, China, October 18–22, 1998. Springer, Heidelberg, Germany. doi:10.1007/3-540-49649-1_16.

[43] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). Cryptology ePrint Archive, Report 2022/975, 2022. https://eprint.iacr.org/2022/975.

[44] André Chailloux, María Naya-Plasencia, and André Schrottenloher. An efficient quantum collision search algorithm and implications on symmetric cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 211–240, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-70697-9_8.

[45] Tung Chou and Jin-Han Liou. A constant-time AVX2 implementation of a variant of ROLLO. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):152–174, 2022. doi:10.46586/tches.v2022.i1.152-174.

[46] George C. Clark, Jr. and J. Bibb Cain. Error-correction coding for digital communications. 2nd printing, 1982.

[47] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997. doi:10.1109/32.605761.

[48] Don Coppersmith and Adi Shamir. Lattice attacks on NTRU. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 52–61, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany. doi:10.1007/3-540-69053-0_5.

[49] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-56880-1_12.

[50] Thomas Debris-Alazard, Léo Ducas, and Wessel P. J. van Woerden. An algorithmic reduction theory for binary codes: LLL and more. *IEEE Transactions on Information Theory*, 68(5):3426–3444, 2022. doi:10.1109/TIT.2022.3143620.

[51] Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10:74–84, 1977. URL: https://ee.stanford.edu/~hellman/publications/27.pdf.

[52] John D. Dixon. Asymptotically fast factorization of integers. *Mathematics of Computation*, 36:255–260, 1981. doi:10.2307/2007743.

[53] Léo Ducas, Maxime Plançon, and Benjamin Wesolowski. On the shortness of vectors to be found by the ideal-SVP quantum algorithm. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 322–351, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26948-7_12.

[54] Il'ya Isaakovich Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25(1):24–32, 1989.

[55] Andre Esser and Emanuele Bellini. Syndrome decoding estimator. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 112–141. Springer, 2022. doi:10.1007/978-3-030-97121-2_5.

[56] Andre Esser and Alexander May. Better sample—random subset sum in $2^{0.255n}$ and its impact on decoding linear codes. 2019. withdrawn. URL: https://arxiv.org/abs/1907.04295.

[57] Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break - solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 433–457, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-07082-2_16.

[58] Agner Fog. Instruction tables, 2023. URL: https://agner.org/optimize/instruction_tables.pdf.

[59] Electronic Frontier Foundation. *Cracking DES: secrets of encryption research, wiretap politics & chip design.* O'Reilly, 1998.

[60] Heiner Giefers and Marco Platzner. An fpga-based reconfigurable mesh many-core. *IEEE Trans. Computers*, 63(12):2919–2932, 2014. doi:10.1109/TC.2013.174.

[61] Oded Goldreich. *Computational complexity: a conceptual perspective.* Cambridge University Press, 2008.

[62] Ian Grigg and Peter Gutmann. The curse of cryptographic numerology. *IEEE Security & Privacy*, 9(3):70–72, 2011.

[63] Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. Cryptology ePrint Archive, Report 2013/162, 2013. https://eprint.iacr.org/2013/162.

[64] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics. Second Series*, 193(2):563–617, 2021. doi:10.4007/annals.2021.193.2.4.

[65] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. doi:10.1109/TIT.1980.1056220.

[66] Martin E. Hellman, Whitfield Diffie, Paul Baran, Dennis Branstad, Douglas L. Hogan, and Arthur J. Levenson. DES (Data Encryption Standard) review at Stanford University, 1976. URL: https://web.archive.org/web/20170420171412/www.toad.com/des-stanford-meeting.html.

[67] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-13190-5_12.

[68] Thomas R. Johnson. *American cryptology during the cold war, 1945–1989, book III: retrenchment and reform, 1972–1980.* 1998. URL: https://archive.org/details/cold_war_iii-nsa.

[69] Elena Kirshanova. Re: Number of bit-operations required for information set decoding attacks on code-based cryptosystems?, 2021. URL: https://crypto.stackexchange.com/a/92112.

[70] Donald Ervin Knuth. *The art of computer programming, Volume III: Sorting and Searching, 2nd Edition.* Addison-Wesley, 1998. URL: https://www.worldcat.org/oclc/312994415.

[71] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 3–22, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-47989-6_1.

[72] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America*,

volume 9230 of *Lecture Notes in Computer Science*, pages 101–118, Guadalajara, Mexico, August 23–26, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-22174-8_6.

[73] Julien Lavauzelle, Matthieu Lequesne, and Nicolas Aragon. Syndrome decoding in the Goppa-McEliece setting, 2023. URL: https://decodingchallenge.org/goppa.

[74] Jonathan D. Lee and Ramarathnam Venkatesan. Rigorous analysis of a randomised number field sieve. *Journal of Number Theory*, 187:92–159, 2018. doi:10.1016/j.jnt.2017.10.019.

[75] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany. doi:10.1007/3-540-45961-8_25.

[76] David P. Leech and Michael W. Chinworth. The economic impacts of NIST's data encryption standard (DES) program, 2001. URL: https://csrc.nist.gov/publications/detail/white-paper/2001/10/01/the-economic-impacts-of-nist-des-program/final.

[77] Hendrik W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics. Second Series*, 126:649–673, 1987. URL: semanticscholar.org/paper/307ab08c3d4f551019297d2480597c614af8069c, doi:10.2307/1971363.

[78] Hendrik W. Lenstra, Jr. Algorithms in algebraic number theory. *Bulletin of the American Mathematical Society. New Series*, 26(2):211–244, 1992. doi:10.1090/S0273-0979-1992-00284-7.

[79] Hendrik W. Lenstra, Jr. and Carl Pomerance. A rigorous time bound for factoring integers. *J. Am. Math. Soc.*, 5(3):483–516, 1992. URL: hdl.handle.net/1887/2148, doi:10.2307/2152702.

[80] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354–1359, 1988.

[81] Gaëtan Leurent and Clara Pernot. New representations of the AES key schedule. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 54–84, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-77870-5_3.

[82] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-25385-0_6.

[83] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-46800-5_9.

[84] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology, January/February 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[85] Dustin Moody. The beginning of the end: the first NIST PQC standards, 2022. URL: https://nist.pqcrypto.org/foia/20220914/pkc2022-march2022-moody.pdf.

[86] Pieter Moree. *Psixyology and diophantine equations*. Leiden: Rijksuniversiteit te Leiden, 1993.

[87] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-45146-4_6.

[88] National Security Agency. NSA's key role in major developments in computer science, 2007. partially declassified in 2017. URL: https://web.archive.org/web/20230430105513/https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/nsa-early-computer-history/6586785-nsa-key-role-in-major-developments-in-computer-science.pdf.

[89] National Security Agency. Yes, we ARE the largest employer of mathematicians in the world, 2014. URL: https://archive.ph/hMV9d.

[90] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. URL: https://doi.org/10.1515/JMC.2008.009.

[91] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[92] Christos H. Papadimitriou. Computational complexity, 1994.

[93] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 685–716, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-17656-3_24.

[94] René Peralta. Circuit minimization work, 2020. URL: http://cs-www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html.

[95] Ray Perlner. Number of bit-operations required for information set decoding attacks on code-based cryptosystems?, 2021. URL: https://crypto.stackexchange.com/q/92074.

[96] Nicole Perlroth, Jeff Larson, and Scott Shane. N.S.A. able to foil basic safeguards of privacy on Web, 2013. URL: https://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html.

[97] Christiane Peters. Information-set decoding for binary codes, 2008. URL: https://github.com/christianepeters/isdf2/.

[98] John M. Pollard. Theorems on factorization and primality testing. *Proceedings of the Cambridge Philosophical Society*, 76:521–528, 1974.

[99] John M. Pollard. A Monte Carlo method for factorization. *BIT. Nordisk Tidskrift for Informationsbehandling*, 15:331–334, 1975. doi:10.1007/BF01933667.

[100] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32:918–924, 1978. doi:10.2307/2006496.

[101] Carl Pomerance. Analysis and comparison of some integer factoring algorithms. Computational methods in number theory, Part I, Math. Cent. Tracts 154, 89–139, 1982.

[102] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[103] Charles M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.

[104] Nathalie Revol and Fabrice Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable computing*, 11(4):275–290, 2005.

[105] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. URL: citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.2023, doi:10.1145/359340.359342.

[106] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin E. Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 241–270, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-70697-9_9.

[107] J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[108] Tarinder Sandhu. Review: AMD Epyc 7742 2P Rome server, 2019. URL: https://web.archive.org/web/20211104084321/https://hexus.net/tech/reviews/cpu/133244-amd-epyc-7742-2p-rome-server/?page=2.

[109] Claus P. Schnorr and Hendrik W. Lenstra, Jr. A Monte Carlo factoring algorithm with linear storage. *Mathematics of Computation*, 43:289–311, 1984. doi:10.2307/2007414.

[110] Claus-Peter Schnorr and Adi Shamir. An optimal sorting algorithm for mesh connected computers. In *18th Annual ACM Symposium on Theory of Computing*, pages 255–263, Berkeley, CA, USA, May 28–30, 1986. ACM Press. doi:10.1145/12130.12156.

[111] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[112] Adi Shamir. Factoring numbers in $O(\log n)$ arithmetic steps, 1977. MIT LCS TM-91. URL: https://web.archive.org/web/20230430125359/https://apps.dtic.mil/sti/pdfs/ADA047709.pdf.

[113] Joseph H. Silverman. Variation of periods modulo p in arithmetic dynamics. *The New York Journal of Mathematics*, 14:601–616, 2008.

[114] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988. doi:10.1007/BFb0019850.

[115] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[116] Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of AES. In Ernest Foo and Douglas Stebila, editors, *ACISP 15: 20th Australasian Conference on Information Security and Privacy*, volume 9144 of *Lecture Notes in Computer Science*, pages 39–56, Brisbane, QLD, Australia, June 29 – July 1, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-19962-7_3.

[117] Clark D. Thompson and H. T. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20(4):263–271, 1977. doi:10.1145/359461.359481.

[118] Andrei L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics Doklady*, volume 3, pages 714–716, 1963.

[119] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 144–161, Fukuoka, Japan, February 24–26, 2016. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-29360-8_10.

[120] U.S. Congress, Office of Technology Assessment. A history of the Department of Defense Federally Funded Research and Development Centers, 1995. URL: https://www.princeton.edu/~ota/disk1/1995/9501/9501.PDF.

[121] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem (keynote talk). In Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu, and Duncan S. Wong, editors, *ASIACCS 11: 6th ACM Symposium on Information, Computer and Communications Security*, pages 1–9, Hong Kong, China, March 22–24, 2011. ACM Press.

[122] Shimeng Yu. *Semiconductor Memory Devices and Circuits*. CRC Press, 2022.

[123] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*, volume 8282 of *Lecture Notes in Computer Science*, pages 29–47, Burnaby, BC, Canada, August 14–16, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-43414-7_2.

A The importance of cryptographic security-level assessments

Cryptography is full of numerical claims regarding costs of attacks against various cryptosystems: attack A costs 2^{56}, attack B costs 2^{80}, attack C costs 2^{112}, etc. This appendix reviews various examples of how these numbers are used, and highlights an attack strategy that sabotages real-world cryptography by exploiting quantitative inaccuracies in evaluations of the costs of attack algorithms.

A.1. The Data Encryption Standard. DES was standardized in 1977, remained an official U.S. government standard until 2005, and was widely deployed in the meantime (see [76]). Recorded DES ciphertexts are now breakable at very low cost, and presumably include some plaintexts that remain useful to attackers today: for example, there is no evident time limit on the type of extortion described in [40].

Diffie and Hellman objected at the outset to the low DES security level. In [51], they explained how to build a $20000000 machine breaking one DES key per day with a brute-force attack. (This was before Hellman [65] introduced much more efficient attacks after precomputation.) NSA claimed that the attack was actually 30000 times more expensive (see [66]: "instead of one day he gets something like 91 years"). The dispute here was not about the number of DES keys (namely 2^{56}), but about lower-level circuit details and claimed overheads (again see [66]; e.g., "for the pipelining you blew up the number of gates, and your size of your chip went up, and the cost went up"). Proposals to use larger keys were explicitly rejected on the basis of (1) the claimed cost of DES attacks and (2) the claimed cost of those proposals; see [1].

The actual cost of brute-force attacks against DES was, in fact, far below NSA's public claims, but public researchers did not have the resources to demonstrate this at the time. Twenty years later, EFF built a $250000 machine [59] breaking one DES key every few days with a brute-force attack (and larger-scale attacks would have had even better price-performance ratios because of various economies of scale), whereas combining NSA's claims with the observed improvements in chip technology over the same period would have predicted three orders of magnitude higher costs.

These three orders of magnitude mean that attackers were, for any particular attack cost, able to break three orders of magnitude more data than NSA was claiming—or break the same data for the same cost many years earlier, with three orders of magnitude worse chip technology. All of this is with simple enumeration of all keys, not using the faster attack from [65].

A.2. Sabotaging cryptographic selection. NSA had secretly established a policy of sabotaging cryptography to make sure it was "weak enough" for NSA to break, in the words of an internal NSA history book [68]: "Narrowing the encryption problem to a single, influential algorithm might drive out competitors, and that would reduce the field that NSA had to be concerned about. Could a public encryption standard be made secure enough to protect against everything but a massive brute force attack, but weak enough to still permit an attack of some nature using very sophisticated (and expensive) techniques?"

Forty years later, the budget for NSA's "Sigint Enabling Project"—a project to "covertly influence and/or overtly leverage" deployed designs to make them "exploitable"—was "more than $250 million a year", according to [96].

Cryptographic choices are continually made on the basis of quantitative claims regarding attack costs. Small-sounding differences in claims played a clear role in, e.g., the DES example above and the much newer NTRU-Kyber example below. Current practice does not follow robust mechanisms to recognize errors in these quantitative claims; it does not even enforce clear definitions of the claims. This gives attackers tremendous freedom to manipulate the numbers, for example by publicly pointing out an overestimate for cryptosystem X while staying quiet about an overestimate for cryptosystem Y.

Sometimes cryptographic systems are publicly broken by attack algorithms that are so fast that academics can easily run them—but then those systems are no longer considered for deployment; the targets of sabotage are the remaining algorithms. In the opposite direction, sometimes cryptographic systems have such high security levels that making them sound breakable would require a gigantic, hopefully obvious, security-evaluation error—but there is tremendous pressure to reject these systems in favor of systems with smaller security margins; see, e.g., [62] and [11]. Because of these incentives, as noted in [22], most security levels end up closely packed—not far from each other, and not far beyond what they need to resist attack.

A.3. NTRU-509 vs. Kyber-512. The following cryptosystem proposals aim for much higher security levels than DES, as one would expect given that (1) DES was a security disaster, (2) these proposals are 40 years newer than DES, and (3) these proposals are portrayed as being secure for many further years into the future. Specifically, these proposals aim to be as secure as AES-128. The point of the following text is not to say that security problems are known in these proposals, but to give an example of the role that a tiny difference in quantitative security claims played in a modern cryptographic standardization decision.

NTRU-509 is the most efficient proposal displayed in [85, page 23, "bandwidth graph"]: in particular, it visibly beats Kyber-512 in that graph. This is a size comparison from a March 2022 NIST talk "The beginning of the end: the first NIST PQC standards". NTRU-509 has 699-byte public keys and 699-byte ciphertexts, while Kyber-512 has 800-byte public keys and 768-byte ciphertexts.

However, NTRU-509 is not present in [7, Table 6], a size comparison from a July 2022 NIST report. NTRU is instead represented in the table by NTRU-677, which has 930-byte public keys and 930-byte ciphertexts, obviously beaten by Kyber-512.

The report announced that NIST would standardize Kyber for post-quantum encryption. The report includes a paragraph on NIST's "difficult" choice between Kyber and NTRU. This paragraph describes Kyber's security assumptions as "marginally" more convincing [7, page 18] than NTRU's security assumptions, but the same paragraph says that "NIST is confident in the security that each provides", so this does not appear to have been a very important decision criterion. The only decisive-sounding sentence in the paragraph is the last sentence: "With regard to performance, Kyber was near the top (if not the top) in most benchmarks."

Why did NTRU-509 disappear between [85] and [7]? According to [7, page 39], "what NIST used for NTRU in the figures and tables in this report" is a "non-local cost model" for the "assignment of security categories". In context, this indicates that NIST eliminated NTRU-509 as not reaching NIST's minimum allowed "security category", namely the security level of AES-128.

NTRU-509 uses lattice problems that are not much smaller than the Kyber-512 lattice problems. For example, according to the Kyber-512 security-estimation methodology, Kyber-512 is just 12 bits harder to break than NTRU-509. This is compatible with NIST concluding that

- NTRU-509 is easier to break than AES-128 and
- AES-128 is easier to break than Kyber-512,

but it forces both of these gaps to be very small, with total just 12 bits; at least one of the gaps must have been 6 bits or smaller.

The decision structure displayed in [7] is thus sensitive to very small changes in quantitative security levels. If algorithm analyses had been modified to produce slightly higher security claims (for example, accounting for missed overheads) then NTRU-509 would not have been eliminated; NTRU would have scored much better in the quantitative comparisons in [7], such as [7, Table 6]. This would not necessarily have been a decisive win for NTRU, but if a choice is labeled as "difficult" then one has to presume that any big change is important. In the opposite direction, if security claims had been slightly lower then Kyber-512 would have been eliminated, and NTRU-677 would have beaten the smallest remaining Kyber option.

A.4. The dangers of underestimates. There is a common perception that underestimating attack exponents by 5% or 10% or 20% is harmless, even "conservative": users will simply choose larger cryptosystem parameters to compensate, ending up with an extra security margin.

However, cryptosystems are continually being compared. Users ask for the highest-performance cryptosystem that fits specified security requirements, or for the safest cryptosystem that fits specified performance requirements. Underestimating security levels corrupts these comparisons, and can mean that a higher-security cryptosystem is skipped in favor of a lower-security cryptosystem. The misevaluation would not matter here if it were the same across cryptosystems, but known examples of underestimates vary in many ways across cryptosystems.

If underestimates are encouraged, or at least accepted, then an attacker trying to influence cryptosystem selection is free to publicly underestimate the security of cryptosystem X more than the security of cryptosystem Y. This is analogous to the attacker's freedom to selectively point out overestimates, but in that context there was a risk (from the attacker's perspective) of a non-attacker discovering and pointing out the other overestimates, which presumably would then be corrected.

A.5. The process of attack discovery. The process of analyzing the costs of an attack is one component of a broader public process of searching for the best attacks.

One might think that errors in attack analyses do not damage this broader process. For example, the subset-sum algorithm of [67], with exponent $0.337n$, was a breakthrough compared to the $0.5n$ exponent that had been known for decades. The ideas of [67]—and of [15], whose main result improved 0.337 to 0.291—were then successfully adapted to a larger class of code/lattice problems. For example, the aforementioned MMT paper [82] adapted [67] to decoding, and the BJMM paper [18] adapted [15] to decoding. Why does it matter that the exponent from [67] was originally underestimated by 8%, or that the exponent from [56] was underestimated by 12%?

If every algorithm speedup were like [67] in changing exponents by more than 30%, then it is true that the speedups would not be hidden by errors around 10%. However, the reality is that, for well-studied problems, such large changes rarely happen all at once. Most individual speedups change exponents by much less than 10%.

For example, [34] reported subset-sum exponent 0.283, just 2.7% better than the 0.291 from [15]. This speedup would have been harder to publish if the erroneous 0.255 from [56] had not been withdrawn.

Small speedups rarely make the news. They are nevertheless the main driver of algorithmic advances, both through the cumulative impact of many small speedups and through the role of small speedups as inspiring larger speedups (e.g., [103] inspiring [64], and [98] inspiring [77]). Underestimating a state-of-the-art exponent by 10% will lead one useful idea after another to be misevaluated as unproductive, and can even halt research on a topic—unless someone luckily happens to find a big speedup *or* discovers that the claimed exponent was wrong.

Now consider the hypothesis that a large-scale attacker has found an algorithm to break a popular cryptosystem. The attacker will then want to keep this knowledge secret—for example, by attracting public cryptanalysis to less productive lines of attack. Underestimating attack costs is a straightforward way to do this.

This hypothesis is plausible. NSA says [89] it is "the largest employer of mathematicians in the world". The U.S. government also funds "federal research centers" that are "designed to attract the best and the brightest people available using salary above the wage scale the federal government offers", according to [120]; in particular, NSA contractor IDA has hired many cryptanalysts, such as Coppersmith, whose pre-IDA papers earned the 2022 Levchin Prize for "foundational innovations in cryptanalysis". As a historical matter, many cryptosystems, including deployed cryptosystems, have been publicly broken, so it is easy to imagine some currently deployed cryptosystems being breakable.

Of course, taking steps to eliminate errors does not guarantee that the public search for the best attacks will succeed. State-of-the-art cryptanalysis is challenging even when the process is not under attack.

B The tension between proving attack costs and optimizing attacks

The literature on algorithms has many examples of speedups that have experimental evidence but no proof. In particular, experience shows that the normal state of affairs for conjecturally hard cryptographic problems is that efforts to attack the problems develop speedups that are conjectured, not proven, to work.

This appendix surveys various examples of this phenomenon for factorization (which has a very long history), elliptic-curve discrete logarithms, and lattice problems. The magnitude of the conjectured speedups varies from one problem to another. These gaps motivate this paper's choice to formalize cryptanalytic processes that are not proofs.

Perhaps all of these speedups will eventually be proven (or disproven), but perhaps not; perhaps some of them are unprovable. Either way, there are long periods during which no proof is known, and there is tremendous value in formalizing the processes that cryptanalysts are following during these periods.

One exception to the normal state of affairs is that the study of a problem often *begins* with simple provable attacks, such as trial division for factorization. However, in each of the examples below, continued study of the problem produced gaps between the state of the art and the proven state of the art.

Another exception is that often systems are broken by new attacks, and sometimes these new attacks are proven while the superseded attacks were not. Broken systems are not what one would recommend to cryptographic users.

B.1. Why unproven attack speedups should be unsurprising. Mathematicians and theoretical computer scientists typically view proofs as their primary research output, and select research topics that have a high chance of producing proofs. This creates a selection bias towards provable statements—and yet these researchers frequently report encountering statements that they are unable to prove, despite accumulating considerable evidence that the statements are true.

One expects removing this selection bias to create a much larger frequency of such statements. This matches the observed situation in cryptanalysis. Cryptanalysts focusing on the real-world task of analyzing security levels of major cryptosystem proposals do not have the luxury of selecting topics for provability; and cryptanalysts continually develop attack speedups that are experimentally observed to work but that nobody seems able to prove.

For comparison, [3] claims that Gödel's "gremlin"—the fact that there are unprovable truths—is "mostly dormant", and partly attributes this claimed situation to a differently stated selection bias: namely, that "pattern-less parts of math" (apparently meaning statements without proofs) are not "interesting to humans" (apparently meaning people who prioritize proofs).

B.2. Existence of unprovable speedups. For readers who perceive Gödel's results as a provable example of a pervasive problem rather than as a mostly dormant gremlin, the following example makes it even less surprising that there are many unproven algorithm speedups.

Recall that Gödel's second incompleteness theorem states, for a broad class of axiom systems S, that if S is consistent then there is no proof in S of consistency of S. It is then an easy exercise to construct an algorithm speedup with the following property: if S is consistent then S cannot prove the speedup correct even though the speedup is, in fact, correct.

For example, define F as the function that, on input x, returns 1 if x is a proof in S of consistency of S, else 0. The straightforward way to compute F reads through all of x to check, step by step, whether x is a proof in S of consistency of S. The speedup[4] is to simply return 0 without bothering to read x. This speedup is correct by Gödel *if* S is consistent. *Proving* in S that the speedup is correct means proving in S that there is no proof in S of consistency of S, which logically implies proving in S that S is consistent,[5] which, by Gödel again, is impossible if S is consistent.

Note that this is not showing that *every* function F has unprovable speedups, and is not showing that the unproven speedups listed below are unprovable. Formally, restricting attention to concrete problem sizes such as RSA-2048, as in real-world cryptography, also means that one can restrict attention to the finite list of attacks that do not consume more resources than brute-force search, and that establishing the effectiveness of each attack is a finite computation, ergo provable. This raises the question of whether there is a proof of length below, e.g., 2^{100}.

B.3. Factorization. For simplicity, the following comments focus on the problem of factoring "balanced" moduli $n = pq$, where the secret primes p and q are chosen independently and uniformly at random between 2^{b-1} and 2^b. Also, these comments focus on operation counts without considering the costs of memory.

For trial division by all primes between 2^{b-1} and 2^b, the number of primes is $(1/2 + o(1))2^b/\log 2^b$ by the prime-number theorem, where log is the natural logarithm and $o(1)$ is something that converges to 0 as $b \to \infty$. More generally, for each real number θ with $0 < \theta \leq 1$, trial division by all primes between 2^{b-1} and $(1+\theta)2^{b-1}$ involves $(\theta/2+o(1))2^b/\log 2^b$ divisions and succeeds with probability $(2-\theta)\theta + o(1)$. One can make the $o(1)$ bounds explicit to prove reasonably precise bounds for concrete values of b; see generally [107].

Some simple speedups to trial division, such as modifying the range of primes considered according to the size of the modulus being attacked, are similarly provable. Even better, in 1974, Pollard [98] introduced an algorithm that provably cuts the exponent in half, in particular reaching probability 1 using just $n^{1/4+o(1)}$ operations.

However, Pollard noted that an earlier method of Shanks was conjectured to use just $n^{1/5+o(1)}$ operations. Furthermore, Pollard's

[4] This example works with a broad class of "speedup" definitions: all one needs is that simply returning 0 without reading x is a speedup compared to checking each step of x.

[5] If S is inconsistent then there is a proof in S of every statement. In particular, if S is inconsistent then there is a proof in S of consistency of S. Take the contrapositive: if there is no proof in S of consistency of S, then S is consistent. Reflect into a proof in S: if there is a proof in S that there is no proof in S of consistency of S, then there is a proof in S that S is consistent.

paper introduced another factorization method, the $p-1$ method, which has its own gap betwen proven effectiveness and conjectured effectiveness.

It is not difficult to formulate a precise quantitative statement that, qualitatively, says that the $p-1$ method finds p quickly if $p-1$ happens to factor into small primes (and similarly for q); what is difficult is to provably pin down the chance that this occurs. There were already bounds on the number of integers between $2^{b-1}-1$ and 2^b-1 that factor into small primes (see [86] for a survey), but this gives only a (conjecturally) loose upper bound on the chance that $p-1$ is one of those integers, and does not give a lower bound.

The proof challenges deepened the next year with Pollard's introduction [99] of the rho method, which iterates a polynomial function such as $x \mapsto x^2+1$ modulo n and hopes to encounter a collision modulo p (or modulo q). The rho method *conjecturally* uses $n^{1/4+o(1)}$ operations for worst-case primes p, q. A closer look shows that the conjectural number of operations is better by a factor $b^{\Theta(1)}$ than the number of operations in the provable algorithm of [98]. What has been proven about the rho method is much weaker than what has been conjectured; see, e.g., [12], [113, Section 6], and [20, Section 5].

The RSA paper then appeared [105], and mentioned, among other things, that a new algorithm from Schroeppel "can factor n in approximately $\exp\sqrt{\ln(n)\cdot\ln(\ln(n))}$" operations. A correct statement of what was known would have said that this was a *conjecture*, would have included exponent $1+o(1)$, and would have stated that the operation count was ignoring linear algebra. A full operation count, including linear algebra, produced a larger exponent, which was brought much closer to 1 by a subsequent change from Schroeppel's linear sieve to Pomerance's quadratic sieve; see [101] for this analysis. Subsequent improvements in linear algebra brought the exponent to $1+o(1)$. All of these exponents are conjectural; the effectiveness of the linear sieve and of the quadratic sieve remains unproven today.

In 1981, Dixon [52] introduced another factorization method, the random-squares method, proven to take $\exp\sqrt{\Theta(\log n \log\log n)}$ operations. Further work improved the Θ constant in the exponent, culminating in a 1992 Lenstra–Pomerance algorithm *proven* to take $\exp\sqrt{(1+o(1))\log n \log\log n}$ operations—but by that time other algorithms had been introduced with much better *conjectural* scalability. As stated in [79, page 484]:

> With our theorem, we hoped to bridge the gap between rigorously analyzed factoring algorithms and heuristically analyzed factoring algorithms. Our victory has turned out to be an empty one, however, since in 1989 factoring broke through the $L_n[\frac{1}{2},1]$ barrier in a rather dramatic fashion.

The breakthrough factorization algorithm mentioned there, the number-field sieve (NFS), has some components that have been rigorously analyzed but other components that remain unproven today. The "rigorous analysis" from [74] of a variant of NFS is actually "conditional on Conjecture 7.1"; see [74, Theorem 2.3]. Even if some variant of NFS is proven to work someday, it seems unlikely that this variant will include all the known NFS speedups: for example, the partial proof in [74] relies on making a random choice of number fields within a large range, while NFS speed records rely on searching for number fields that appear particularly favorable.

In reasonable models of *quantum* computation, Shor's algorithm has much better scalability than NFS, and at the same time is provable. Shor's algorithm is normally interpreted as a reason *not* to use RSA. "Post-quantum RSA" [26] instead scales RSA up to sizes that resist Shor's algorithm; security analysis of this RSA variant relies on analysis of how well Lenstra's elliptic-curve method [77] performs—which is yet another conjecture that remains unproven decades later.

B.4. Elliptic-curve discrete logarithms. In the case of discrete-logarithm algorithms for conservative choices of elliptic curves (not, e.g., pairing-friendly curves), speedups have been quantitatively much smaller than for factorization algorithms. However, there are still gaps between the best proven effectiveness of known algorithms and the best conjectured effectiveness of known algorithms.

For example, Pollard [100] introduced a rho method for discrete logarithms (and a "kangaroo" method for an important variant of the same problem, namely discrete logarithms in a short interval). The rho method uses much less memory than the baby-step-giant-step method of computing discrete logarithms, and the number of operations is *conjecturally* within a small constant factor of the proven number of operations of the baby-step-giant-step method.

This rho method does not need to follow the polynomial structure that was used in the rho method for factorization. A *provable* variant of the rho method computes $\log_P Q$, where P and Q are curve points, by walking from R to $a(R)P+b(R)Q$ for functions a, b chosen uniformly at random. However, this variant has to keep building and checking a table showing the $a(R), b(R)$ values chosen so far; this throws away the main advantage of the rho method, namely that it uses very little memory.

Conjecturally optimized versions of the rho method instead walk from R to $R+W_{H(R)}$, with further modifications to exploit fast negation on elliptic curves. Here H is an extremely lightweight hash function that hashes R to just a few bits, and the values $W_0, W_1, \ldots$ are chosen as random linear combinations of P, Q. The group structure does not interact in a problematic way with the structure of H in experiments, but this is a heuristic, not a proof. The analysis of adding a small number of values involves more heuristics; see generally [28].

B.5. Lattice problems. An SVP algorithm with *proven* exponent $1+o(1)$ was introduced by Aggarwal–Dadush–Regev–Stephens-Davidowitz [5] in 2015. For comparison, Nguyen–Vidick [90] had already obtained *conjectural* exponent $0.415\ldots+o(1)$ in 2008. Attacks after [90] obtained conjectural exponents close to 0.384 [121], 0.3778 [123], 0.3774 [17], 0.337 [71], 0.298 [72], and 0.292 [16].

The application of SVP algorithms inside lattice attacks involves additional heuristics. For example, deployed lattice-based cryptosystems essentially always rely on lattices derived from number fields, but analyses of the performance of BKZ in this context generally ignore this lattice structure. See, e.g., [111, Section 5.1, "analyze the hardness of the MLWE problem as an LWE problem"]. Nothing has been proven here; treating structured lattices as unstructured lattices is another heuristic.

Some attacks explicitly use the number-field structure. The analyses rely on further heuristics: see, e.g., [93, Heuristics 1–6]. This

follows a long-established pattern of relying on heuristics in analyzing algorithms for number fields. Consider, for example, the following 1992 comment from Lenstra [78]:

> 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].

This is also one of the reasons that heuristics appear in the analyses of various factorization algorithms listed above—not just the number-field sieve.

C Examples of errors that comprehensive formalization would have prevented

This appendix looks at why the errors in [67] and [53] mentioned in Section 1 were not caught by the simulations in those papers. In each case, there was a mismatch at one of the interfaces between the simulations and the incorrect analysis, and there was nothing checking for errors hiding in that interface, whereas the formalization pattern stated in Section 1.2 would have enforced a check. The two errors illustrate mismatches at two different interfaces.

C.1. A mismatch of models of computation. For [67], heuristic analysis missed a bottleneck inside the algorithm. Using the analysis to compute a cost formula, and directly comparing this formula at any particular input size to the algorithm's actual cost, would immediately have detected the discrepancy. This would have required [67] to specify a model of computation, to track costs in that model through the analysis, and to measure the algorithm's cost in the same metric.

Instead the analysis in [67] tracked asymptotics of costs in an unspecified model of computation. Meanwhile the attack experiments in [67] tracked measurements of run time on a real CPU. The hardware design that produces a real CPU is a specified model of computation, but the definition is very complicated and obviously not the foundation of the analysis in [67]. Consider the fact that the analysis assumes instantaneous access to arbitrarily large arrays, while real CPUs vary heavily in array-access costs depending on the size of the array; see, e.g., the measurements in [2, "Memory Access and Caching"].

This mismatch of models made meaningful comparison impossible. Trying experiments for enough sizes might have detected that the attack had worse scalability than $2^{0.3113n}$, but this would have been easily explainable in other ways, such as $2^{o(n)}$ factors suppressed in the analysis and memory-access costs visible on the CPU.

C.2. A mismatch of problem parameters. Cost was not an issue for [53]: that paper considered only algorithms having low cost (asymptotically bounded by a low-degree polynomial). The issue was identifying the transition between problems solved with low probability and problems solved with high probability.

The lattice problem attacked in [53] is conventionally indexed by "Hermite factor", the simplest quantity to use in checking alleged solutions. Internally, the paper's analysis introduced another quantity, the Hermite factor times "$\Delta_K^{1/2n}$". The simulation in [53] also worked with that quantity. The (unpublished) software producing the erroneous Hermite-factor graph in [53] should have divided the internal quantity by "$\Delta_K^{1/2n}$", but instead multiplied by "$\Delta_K^{1/2n}$". The way this error was detected two years later was by another team redoing the entire sequence of computations outlined in [53], including production of the graph.

Formalization of the problem would have used Hermite factor, for the same reason that Hermite factor is conventionally used. The simulated algorithm output would have been compared directly to the Hermite factor. The success probability of the algorithm would have been compared to the formula predicting success probability for the same Hermite factor. A graph automatically generated from the same formula would never have been faced with a different quantity. If, as a variant of the error in [53], division and multiplication had been exchanged inside the formula, then this would have been caught by the probability comparisons.

C.3. The value of post-mortems. Studying how errors occurred is useful for evaluating the benefits of error-detection techniques. This process requires not merely acknowledgment of errors, but also analysis of how the errors occurred and analysis of how the errors could have been prevented.

Beyond an initial set of post-mortems, further post-mortems can be useful for identifying further types of errors. For example, the error in [109] was an error in the probability analysis for *some* inputs, and would have been caught by systematic experiments for small inputs; [109] included some experiments, but only for larger inputs, which were less likely to trigger the error. This shows the importance of checking many inputs, including many small inputs. The situation of the error in [109] is different from the situations of the errors in [67] and [53], where a precise check of *one* input would have been enough.

Note that there is an inherent selection bias in studying only *known* errors. It is important for post-mortems to be accompanied by analyses of how further types of errors can occur and can be caught.

D Breaking security claims for Kyber-512 and AES-128

This appendix reviews, and disproves, two examples of claims regarding the number of "gates" used by "optimised"/"optimal" attack algorithms. There are problematic ambiguities in both claims, but the disproof here is reasonably robust against these ambiguities:

- The claimed security levels are not accompanied by clear definitions of the allowed set G of "gates", but the claims are accompanied by statements making sufficiently clear that particular "gates" are *included* in G. The attacks in this appendix are built purely from those "gates".
- The claimed security levels are accompanied by ambiguous wording regarding precision ("about" and "estimate"), but it is not plausible that most readers would interpret

> the wording as allowing the magnitude of attack speedups demonstrated in this appendix.

The big problem with the claims has nothing to do with these ambiguities. Both claims use a concept of "gate" in which access to an arbitrarily large array costs just 1 "gate". This appendix presents very easy ways to exploit this low-cost array access.

In particular, the latest Kyber documentation [111] states a "cost of about $2^{137.4}$ gates for AllPairSearch in dimension 375" for the main subroutine used inside a Kyber-512 attack, and NIST [91] states an "estimate" of "2^{143} classical gates" for "optimal" AES-128 key search. For Kyber-512, this appendix cuts almost 10 bits out of the "gate" count for the "primary optimisation target", and also speeds up various secondary algorithm components. The AES-128 speedup is smaller but is still sufficient to disprove 2^{143}.

There have been several other Kyber-512 attack speedups after [111] (although there also appear to be ongoing disputes regarding the success probability of some of those speedups). All of those appear to combine straightforwardly with the speedups here.

The speedups in this appendix as measured by "gates" are slowdowns in realistic cost metrics. The point here is not merely that this "gates" concept is unrealistic, but also that the algorithm optimization in the relevant literature obviously did not focus on this notion. See Appendix D.5 and Appendix G.5 for connections to the question of how models of computation and cost metrics should be selected.

D.1. Which definition of "gates" is being used? In its 2016 call for submissions [91] to the NIST Post-Quantum Cryptography Standardization Project, NIST gave "estimates for the classical and quantum gate counts for the optimal key recovery and collision attacks on AES and SHA3"—but did not define the allowed set of "gates".

In particular, NIST estimated "2^{143} classical gates" for AES-128 key search, and designated AES-128 key search as the minimum security level allowed in the project. There were then various requests for a definition of the allowed set of "gates". In a 2022 report [7], NIST wrote the following:

> In the context of the NIST PQC Standardization Process, the version of the RAM model, where 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, is sometimes referred to as the *gate count* model.

This is still not a complete definition—one can write down many different models that fit all of the features listed here for "the" model—but the statement that "each one-bit memory read or write is counted as one bit-operation" is sufficient for the AES-128 attack speedups in this appendix.

The same report [7, page 18] highlights the "thorough and detailed security analysis" in the round-3 Kyber specification. That specification, in turn, estimates [111, page 27] that attacking Kyber-512 involves $2^{151.5}$ "gates": specifically, $2^{14.1}$ calls to "AllPairSearch", times "a cost of about $2^{137.4}$ gates for AllPairSearch in dimension 375". The latter cost is based on "explicit gate counts for the innermost loop operations (XOR-popcounts, inner products)" and is attributed to [9].

The paper [9] says that it describes "classical algorithms as programs for RAM machines (random access memory machines)", and counts the number of "NOT, AND, OR, XOR, LOAD, STORE" operations where "LOAD and STORE act on ℓ bit registers". This is again not a complete definition, but enough information is provided to allow some comparisons to [7].

"NOT, AND, OR, XOR" appear to be intended as 2-input operations, so they are examples of NIST's "act on no more than 2 bits at a time". NIST appears to be allowing other such operations such as NAND, but this makes only a small difference in operation counts.

A more important incompatibility between [9] and [7] is that "LOAD" and "STORE" in [9] have multiple-bit addresses *and* transfer multiple bits of data at once, whereas [7] allows multiple-bit addresses but only a "one-bit memory read or write". This difference is not clear from the brief description "LOAD and STORE act on ℓ bit registers" but can be seen from the analogy stated in [9] to particular quantum "gates"; it can also be seen, more straightforwardly, from the statement "loading $h(v)$ has cost 1" in [9, Section 4.2], where $h(v)$ is an n-bit vector.

The speedups in this appendix are generally larger with multi-bit loads than with single-bit loads, although some of the tables below have single-bit outputs. For breaking the Kyber-512 security levels claimed in [9] and [111], this appendix allows multi-bit loads, since these are allowed and used in [9] and there is nothing to the contrary in [111]; but this appendix also notes what would happen with single-bit loads. For breaking the AES-128 security levels claimed in [91], this appendix restricts to single-bit loads.

Neither [9] nor [7] appears to prevent extremely large tables from being embedded into programs, such as precomputed tables simply mapping public data to secret keys. This appendix limits itself to small attack algorithms building tables at run time, so the speedups here apply even if program length is added into cost as in, e.g., [19].

D.2. Components of the $2^{137.4}$ claim for Kyber-512. The paper [9] says that a "XOR and Population Count" operation, "popcount", is its "primary optimisation target". This operation "loads u and v from specified memory addresses, computes $h(u)$ and $h(v)$, computes the Hamming weight of $h(u) \oplus h(v)$, and checks whether it is less than or equal to k".

The "RAM program for popcount" in [9, Section 4.2] begins by saying that "loading $h(v)$ has cost 1". This illustrates that [9] is allowing and using cost-1 multi-bit memory access, as noted above.

The program then carries out a sequence of bit operations on the bits of $h(u)$ and $h(v)$ to build a tree of adders ending with the Hamming weight. The "overall instruction count is $6n - 4\ell - 5$" where "$\ell = \lceil \log_2 n \rceil$". For example, for dimension $d = 375$, [9] takes $n = 511$, so the "instruction count" is 3025.

As for "inner products", [9, Section 4.3] explains that this does not need careful optimization: "The cost of one inner product is amortised over many popcounts, and a small change in the popcount parameters will quickly suppress the ratio of inner products to popcounts (see Remark 2). Hence we only need a rough estimate for the cost of an inner product." The inner-product cost estimate given in

[9, Section 4.3] is "approximately 32^2d" for d 32-bit multiplications; here 32^2 is the number of ANDs in schoolbook multiplication.

The script in [9] covers many smaller algorithm components that are not commented upon in the text of [9]. A review of these components shows that the number of bits manipulated is continually appearing. For example, the script in [9] estimates cost $(32 + \log_2 Z)Z(\log_2 Z)$ for sorting a list of Z 32-bit integers.

D.3. Exploiting tables to reduce the number of "gates". Consider a table mapping pairs (r, s), where r and s are 54-bit vectors, to the 6-bit Hamming weight of $r \oplus s$. It is easy to build this table using just 2^{110} "gates", which is not a bottleneck in the attack.

Apply this table to the bottom 54 bits of $h(u)$ and $h(v)$, then to the next 54 bits of $h(u)$ and $h(v)$, etc. There are 7 table lookups, reducing the input to 7 Hamming weights, each having 6 bits. Then use one further lookup in another table to map these 42 bits to the desired single-bit output, namely whether the sum "is less than or equal to k".

With the memory access allowed by NIST in [7], this costs just 43 "gates" for the 43 bits of table output. Even better, with the more powerful memory access in [9], this costs just 8 instructions for the 8 table lookups. This is 378 times better than the 3025 instructions used in [9].

Similar comments apply to inner products: precomputing multiplication tables and addition tables easily reduces the cost of d 32-bit multiplications and additions to just $3d$ instructions, almost three orders of magnitude better than the "approximately 32^2d" from [9, Section 4.3]. The speedup is not as large if each output bit is counted as in [7], but one can skip most of the output bits as explained in [9].

Sorting can also easily exploit multi-bit LOAD and STORE. A simple merge sort uses just a few instructions per comparison after precomputation of increment tables, decrement tables, comparison tables, etc. More broadly, essentially every combination of "NOT, AND, OR, XOR" operations in [9] includes long stretches of operations that can be productively replaced with table lookups, given that [9] allows LOAD and STORE as single "gates".

D.4. The AES-128 baseline. NIST has never provided details of how it arrived at its estimate of "2^{143} classical gates" for AES-128 "key recovery".

An AES-128 block encryption involves 10 rounds, each involving 4 S-box lookups to compute a round key and 16 S-box lookups for encryption, for a total of 200 S-box lookups. There were already various efforts to minimize the number of bit operations for the AES-128 S-box: for example, [35] reported "32 AND gates and 83 XOR/XNOR gates for a total of 115 gates", meaning 23000 bit operations for AES-128. Beyond the S-boxes, there are various linear operations plus a final ciphertext comparison; without a precise calculation, it is reasonable to estimate a total close to 2^{15} bit operations. (For a precise calculation, see Appendix I.)

One expects "key recovery" to search only 2^{127} keys on average, not 2^{128}, for a total of about 2^{142} bit operations. A single plaintext-ciphertext pair will, conjecturally, identify just a few possibilities for the key, and checking those few possibilities against a second plaintext-ciphertext pair has negligible extra cost; there is no need to apply every key guess to both plaintexts.

(If iterations were independent, as in the simplest forms of ISD, then 2^{15} operations for an iteration having success probability 2^{-128} would mean an average of 2^{143} operations until success. Iterations in exhaustive AES key search are not independent: a failed key guess increases the success probability of subsequent key guesses.)

More importantly, given that NIST says in [7] that "each one-bit memory read or write is counted as one bit-operation" in "the gate count model", it is easy to reduce AES-128 encryption to far fewer than 2^{15} "gates".

As a starting point, consider a conventional "T-table" implementation. Each of the 10 encryption rounds performs the following operations:

- 16 table lookups for the 16 bytes of state, where each table lookup produces 32 bits of output. This costs 512 "gates".
- XORing each of 128 bits of a round key with 4 of the bits from table lookups. Each XOR of 5 bits costs just 1 "gate" with a XOR-5-bits table, so overall this costs 128 "gates".
- 4 further table lookups for the round key, costing 128 "gates".
- 128 further "gates" for round-key XORs.

Overall this is 896 "gates" for each of the 10 rounds, for a total only slightly above 2^{13} "gates", including comparison of the resulting 128 bits to a given 128-bit ciphertext. Key recovery then takes, on average, slightly above 2^{140} "gates".

One can do even better by building tables that take, e.g., 32 bits of input at once. It is not obvious how far this can be pushed: writing down 1280 bits of state and 1280 bits of round keys requires at least 2560 "gates", but perhaps it is possible to do better by writing down only the nonlinear components of round keys and by merging rounds. This requires analysis of how potential table structures interact with the large-scale data flow in AES, a complication that does not appear in conventional optimization of Boolean circuits for AES.

D.5. Confidence that attacks have been optimized? The way that the above speedups break the claimed security levels for Kyber-512 and AES-128 is not by introducing new attack ideas, but rather by straightforwardly exploiting the declaration that access to a large array has as low cost as a bit operation.

Quotes such as "each one-bit memory read or write is counted as one bit-operation" and "loading $h(v)$ has cost 1" make clear that this declaration was not an accident. Allowing low-cost memory access is often portrayed as a "conservative" way to measure security, supposedly immunizing security analyses to improvements in time-memory tradeoffs.

However, the resulting security-level claims are incorrect. Inspecting how the literature arrived at these claims shows that most components of the algorithm designs and analyses were closer to what one would expect in a conventional Boolean-circuit model. Instead of carefully distinguishing different gate sets and tracing the impact of this difference upon security levels, the literature treated optimizations for a restricted set of "gates" as if those were optimizations for a broader set of "gates".

For comparison, NIST's report [7] includes the following statement:

> Additionally, while some submitters have rightly observed that many widely used cost models, such

> as the RAM model, underestimate the difficulty of certain memory intensive attacks, the comparative lack of published cryptanalysis using more realistic models may bring into question whether sufficient effort has been made to optimize the best-known attacks to perform well in these models.

This statement appears to indicate that attack designers are normally working in "the" RAM model, and systematically taking advantage of low-cost memory access. It is difficult to reconcile this with the AES-128 and Kyber-512 examples.

E Definitions of Boolean circuits and cost metrics

Section 2.1 gives a particular definition of circuits and of circuit cost. This appendix reviews various alternatives.

One can tweak the costs of gates to come closer to reported hardware costs: for example, assigning cost 2/3 for NOT, cost 1 for NAND, cost 1 for NOR, cost 4/3 for AND, etc. Reports vary depending on the circuit technology considered, and in any event this changes costs by at most a small factor. This paper opts for the simplicity of taking cost 1 for each gate beyond constants and copies.

Rather than allowing every function taking at most 2 inputs, the literature typically defines Boolean circuits to use a smaller universal set of bit operations: sometimes just a few commonly used operations (e.g., [61, Section 1.2.4.1] uses just AND, OR, NOT); sometimes just NAND for minimality. Sometimes 0 and 1 are provided as extra inputs rather than operations. Sometimes the circuit designer is instead required to eliminate 0 and 1, making it impossible to compute, e.g., the 0-bit-to-2-bit function $() \mapsto (0, 1)$; for example, [61, page 39, "any Boolean function can be computed by some family of circuits"] is incorrect with the definitions given in [61, Section 1.2.4.1]. Typically the circuit designer is required to eliminate copies, forcing extra operations for computing, e.g., $(x, y, z) \mapsto (x, y, z, z, y)$ and generally complicating circuit composition.

Often many-input AND, OR, and XOR gates are allowed, with cost proportional to the number of inputs. (For example, [61, Section 1.2.4.1] allows many-input gates, and defines circuit "size" as the number of edges; this is different from Section 2.1 and, e.g., [92, Definition 4.4], where each gate allows at most 2 inputs.) These many-input gates can be converted into a chain of 2-input operations at similar cost (or into a tree, but this paper does not try to minimize circuit depth).

Another typical choice in the literature (used in, e.g., [61] and [92]) is to define Boolean circuits as labeled directed acyclic graphs, where the labels indicate how inputs correspond to vertices, how outputs correspond to vertices, and which bit operations are carried out by non-input vertices. This requires additional labeling when asymmetric operations such as $x_k = x_i(1 - x_j)$ ("ANDN") are allowed, but typically each asymmetric operation is decomposed into two symmetric operations, avoiding the issue. Topological sorting converts such DAGs into circuits meeting this paper's definition.

F Validation of the selected model

It is well known that formalized specifications need validation—sanity checks on the utility of what has been specified. Validation should not be confused with the process of formally verifying conclusions within a model.

This appendix evaluates various aspects of the utility of the specific model of computation and cost metric formalized in CAT.

F.1. Special-purpose circuits. Bitcoin-mining ASICs are special-purpose circuits that compute cryptographic hashes much more efficiently than available general-purpose computers. Circuit-design courses explain in detail how to build such circuits, with portions of the circuit area allocated to bit operations and connected by wires.

This is close to the conventional Boolean-circuit model selected in Section 2.1. One difference is that the reported real-world circuit cost is higher for (e.g.) AND than for NAND, although this is a small effect; see Appendix E. A larger difference is that a Boolean-circuit model does not account for the physical layout of bit operations, and in particular does not account for the cost of communicating data through long wires; see Appendix F.6. Boolean-circuit models also unroll computations of any size, whereas real circuits repeatedly apply limited-size computations; such size limits restrict the model of computation and in particular limit memory consumption, although this restriction is conceptually compatible with the iterative structure of typical attack algorithms.

There is extensive literature indicating that special-purpose circuits outperform general-purpose computers for a much wider range of computations, even when size limits and wiring costs are taken into account. Non-recurring chip-engineering costs are large enough to prevent academic demonstrations of most of these circuits, but large-scale attackers have much larger budgets and presumably rely on special-purpose circuits for their most challenging attacks, the same way that Bitcoin mining moved to ASICs.

An internal 2007 NSA document, partially declassified in 2017, stated [88, page 1] that "since the middle of the last century, automation has been used as a way to greatly ease the making and breaking of codes"; that NSA was already building "special-purpose computers" for "cryptanalysis" [88, page 3] in the 1950s; and that NSA "has a great demand for microchips", where the details of this demand for microchips remained classified [88, page 6]. The document concluded as follows: "NSA's computers almost always were well in advance of data processing equipment anywhere else. In conjunction with its partners in industry and academia, NSA continues to be a leader in research and development of computer technologies and has been a singular pioneer on the frontiers of computer science and electrical engineering."

F.2. Formalizing main computations after precomputations. The way that the CAT formalization is structured (see Section 3) requires circuits for any particular parameters to be produced by an algorithm taking the parameters as input, but does *not* place any limits on how long the algorithm takes to run, beyond the user's patience in running the simulator. Consequently, when a precomputation produces a circuit that costs C, the formalization directly measures C. This models the real-world situation that a large-scale attacker designs and builds special-purpose hardware to

efficiently attack a cryptosystem and wants to know how efficient the resulting hardware is.

One might also try to measure the precomputation time, so as to quantify tradeoffs between precomputation time and main-computation time. Beware, however, that someone can carry out the precomputation in advance and embed the output of the precomputation into an algorithm provided to the measurement process, hiding the precomputation time from the measurement process. There is a long history of definitions that were incorrectly believed to solve this problem (see the attacks in [27]), and there continues to be a common misperception that RAM models prevent precomputation (see Appendix G.4). The lack of definitions capturing the intuitive concept of precomputation time is a general limitation in the literature, neither solved nor exacerbated by the choice of a circuit model.

F.3. Formalizing randomized computations. The formalization in CAT supports randomized attacks, even though the model of computation is deterministic. The point here is that the algorithm that computes a circuit is free to generate random bits. If a randomized circuit (with the usual definition: the input bits are supplemented with some random bits) were allowed and achieved success probability p, then there would exist a choice of randomness for which the circuit achieves success probability at least p, so the randomized circuit would be no better than a deterministic circuit using that choice of randomness.

F.4. Formalizing computations with variable costs. The formalization also supports variable-cost attacks (meaning that the cost depends on the input or on randomness or both, not just on parameters), even though the model of computation is constant-cost. For example, the success probability of an I-iteration ISD algorithm inside CAT is, for each I, the same as the success probability that a variable-iteration ISD algorithm finishes using at most I iterations. Varying I, as in this paper's examples, then shows the distribution of the number of iterations needed by the variable-iteration algorithm.

F.5. Bit-operation counts as lower bounds for real-world costs. Consider a real-world attack using hardware of physical mass M. Assume that the hardware has price at least pM for a positive constant p, constant meaning independent of M and the attack details; one should be able to determine p from the technology used for the attack. If the attack runs for time T then its price-performance ratio[6] is at least pMT.

Assume that the hardware performs computation via bit operations; formalizing quantum attacks is out of scope for this paper. Assume that carrying out a bit operation inherently occupies hardware mass at least m and time at least t—or, more to the point, mass-time product at least mt—for some positive constants m and t determined by the technology. The total number of bit operations carried out by the attack then cannot exceed MT/mt.

Counting the number of bit operations is thus putting a lower bound on the mass-time product, and thus the price-performance ratio, of any attack using this technology. Specifically, the mass-time product MT is at least mt times the number of bit operations, and the price-performance ratio is at least pmt times the number of bit operations.

F.6. How close are bit-operation counts to real-world costs? For a wide range of M, sellers are offering Bitcoin-mining ASICs of total mass M for price proportional to M (with a technology-dependent constant), aside from minor discretization effects. The number of hashes per second carried out by these ASICs is also proportional to M (with another technology-dependent constant), so the ASICs have price-performance ratio proportional to the number of bit operations. This does not mean that the constant of proportionality is as low as pmt (consider, e.g., the aforementioned variations in the costs of bit operations), but there is no evident reason for a large gap.

There are other types of computations for which real-world costs are structurally forced to be farther above bit-operation counts, specifically because bit-operation counts ignore the costs of long-distance communication. For example, standard circuit constructions multiply two n-bit integers using $n^{1+o(1)}$ bit operations (see, e.g., [118]), but a theorem from [37] states, for a reasonable model of two-dimensional circuits, that n-bit multiplication cannot have price-performance ratio better than $n^{3/2+o(1)}$. See also [117] for an analogous theorem regarding sorting, a critical subroutine in many algorithms.

One can try to avoid this asymptotic argument by declaring that all real-world computations have cost bounded by a constant, making it formally meaningless to consider asymptotics of real-world computations as n grows. However, asymptotics are merely the simplest way to see the issue highlighted in [37], namely that communicating data across distance d occupies at least d wire elements each for at least one unit of time. Bit-operation counts ignore this cost, while it is not at all clear that costs sublinear in d can be achieved by any physically realizable communication technology. Perhaps there is a way to manage the energy-input and energy-output difficulties of packing multiplication or sorting into an efficient three-dimensional circuit, but this would at best reduce d from the scale of $n^{1/2}$ to the scale of $n^{1/3}$.

These considerations suggest that moving from the model in Section 2.1 to a circuit-layout model, such as the two-dimensional models of [117] and [37] or possibly a three-dimensional model, would gain realism. This would allow full tracking of circuit sizes (not just the portions of circuits designated by algorithm designers as memory) and of long-distance communication costs. The main disadvantage is the complication.

For the case of ISD, this paper's results already show that various high-memory algorithms have only a marginal benefit against the McEliece cryptosystem even when the costs of long-distance communication are ignored. For example, for $n = 3488$, Table 1 lists $2^{150.59}$ operations for high-memory algorithms and $2^{155.38}$ operations for low-memory algorithms. A model incorporating those costs would thus not make much difference for this case study. Accounting for communication costs would be more important for, e.g., lattice-based cryptosystems, where high-memory attacks play a larger role in the literature.

[6] "Price-performance ratio" is standard engineering terminology for the quotient between (1) price measured in whichever price units and (2) performance measured as operations per unit time. In this case, the price is $\geq pM$, and the performance of an attack run is $1/T$ attacks per unit time, so the price-performance ratio of the attack is $\geq pM/(1/T)$, i.e., $\geq pMT$.

F.7. Further validation. Boolean-circuit models are a common feature of computational-complexity textbooks such as [92] and are widely used in the literature. There are some common variations in the details of the definitions (see Appendix E), creating the usual risks from mismatched interfaces. On the other hand, these variations are quantitatively and qualitatively far less severe than common variations in definitions of RAM models; see generally Appendix G.

CAT includes internal tests showing that various simple circuits have, within the formalization, costs matching what a human calculated from the definition in Section 2.1. There have also been human double-checks of the central bit-operation-counting code inside CAT against that definition.

G RAM models

Instead of a simple Boolean-circuit model (as in Section 2.1 or, more broadly, Appendix E), one could select and formalize one of the more complicated random-access-machine models (RAM models) from the literature. This appendix considers various issues raised by this possibility.

G.1. Which RAM model? A Google Scholar search for `"the RAM model" "bits"` currently finds 1830 papers. A random sample from the first 1000 papers finds that a large fraction do not define "the RAM model". Readers of such papers are led to believe that "the RAM model" refers to a standard, fully defined model of computation and accompanying cost metric. However, a closer look at the literature rapidly finds severe definitional problems.

Consider, for example, the textbook [61, pages 25–26] defining a RAM model with "reset", "inc", "dec", "load", "store", and "cond-goto" instructions. This seems reasonably clear at first glance.

The book then says that "to make the RAM model closer to real-life computers, we may augment it with additional instructions that are available on real-life computers" such as "add" and "mult". The reader is invited to add "instructions that are available in some real-life computer". Obviously this is not just one definition: it is a family of definitions, where the more complicated definitions are motivated by the original definition sounding too restrictive.

A reader briefly checking documentation for "real-life" computers would think that it is safe to include addition, subtraction, multiplication, and division instructions. One finds such an instruction set listed in, e.g., the definition in the earlier textbook [6, page 6], which lists "READ" (direct and indirect), "STORE" (same), "LOAD", "ADD", "SUB", "MULT", "DIV", "WRITE", "JUMP", "JGTZ", "JZERO", and "HALT" instructions (while also inviting the reader to add "any other instructions found in real computers").

However, Shamir's algorithm from [112] factors n in $O(\log n)$ "arithmetic steps (addition, subtraction, multiplication and integer division)". The basic problem is that this model allows a single instruction to handle arbitrarily large integers.

Another textbook [92, Section 2.6] defines a RAM model similar to [6] but with "MULT", "DIV", "WRITE", and "JGTZ" replaced with "HALF", "JPOS", and "JNEG" instructions. There are no multiplications; integers in this model grow by at most one bit at each step. This in turn avoids the extreme abuses of [112], as noted in [92, page 38]; [92, Theorem 2.5] says that this RAM model can be simulated in cubic time by a Turing machine. However, the model still allows a program running in "time" T to carry out arithmetic on $\Theta(T^2)$ bits spread across $\Theta(T)$ integers; this is unrealistic, and not suitable for fine-grained algorithm analysis.

One response is to count the number of bits used in each integer; this is stated in [6, page 12] as an option, the "logarithmic cost criterion". A similar response is to restrict the allowed set of arithmetic operations, allowing only bit operations. However, one can still abuse the basic assumption of cost-1 RAM lookups, as illustrated by the attacks in Appendix D. Assigning higher cost to RAM begs the question of what this cost should be. (Note that implementing a RAM circuit on top of bit operations very much as in real hardware, and then counting the bit operations in this RAM circuit, provides a principled answer to the cost question.)

To summarize, "the" RAM model is actually a large, unstable collection of different models, including many abuse-prone models. One could pick a particular RAM model to clearly define and formalize as an extension to CAT, but there is obviously a high risk that the resulting model will warp whatever algorithm analyses are carried out in the model, while at the same time matching very little of the literature.

G.2. Different roles of models of computation. Historically, one of the earliest uses of models of computation was to prove that various models are equivalent in the sense of supporting the same set of computable functions. Later this was refined into proving that various models with accompanying time metrics are equivalent in the sense of supporting the same set of polynomial-time-computable functions. These simplifications are helpful for building the theories of, respectively, computability and polynomial-time computability.

For example, [61] introduces RAM models not to suggest them as a foundation for algorithm analysis, but as evidence for the idea that Turing machines can compute anything that is intuitively computable. Similarly, [92, page 38] says that the $\Theta(T^2)$ issue is "inconsequential" since it is polynomially bounded.

However, algorithm users—including large-scale attackers—care about the gaps between 2^n and $2^{0.5n}$ and $2^{0.5n}/1000$. Cryptography requires accurate analyses of algorithm costs (see Appendix A); selecting an inaccurate model can easily spoil this, even when every algorithm is correctly analyzed within the model. Low-precision equivalences among models are not helpful in this context; one instead has to carefully distinguish different models, and evaluate gaps between the models and reality.

G.3. Are RAM metrics more realistic than circuit metrics? Typical cost metrics for RAM models assign cost 1 to random access, whereas typical cost metrics for circuit models end up counting every bit operation involved in random access, and end up concluding that random access to an n-bit array costs $\Omega(n)$. This quantitative gap directly affects analyses of a wide range of algorithms.

Introductory algorithm courses teach students to count instructions and label the result as "time", in particular with random-access instructions taking "time" 1. This creates a perception that $\Omega(n)$ is an overestimate of the cost of random access. Students might later learn that 1 is an underestimate of real time—measurements of n-bit random-access time on real CPUs follow roughly a square-root curve as n grows (see, e.g., [2]), as one would expect from the

two-dimensional models cited in Appendix F.6—but still think that $\Omega(n)$ is an overestimate.

However, these time measurements hide a much more important cost of random access: namely, randomly accessing a real n-bit RAM circuit occupies the entire circuit for a moment (see [122, Section 1.3]), for an $\Omega(n)$ price-performance ratio. This is a special case of the fact that bit-operation counts put lower bounds upon price-performance ratio of all computations; see Appendix F.5.

Consequently, typical RAM metrics are, contrary to the above perception, farther from the price-performance ratio of random access than typical circuit metrics are.

For comparison, the same mass of circuitry running a parallel computation for the same time could have been used to carry out $\Omega(n)$ bit operations and thus, e.g., $\Omega(n)$ separate hash computations (with a smaller Ω constant, reflecting the cost of each hash computation), as illustrated by the Bitcoin-mining ASICs mentioned in Appendix F. It would be very strange to use a cost metric that assigns cost $o(n)$ to $\Omega(n)$ separate hash computations.

Array access becomes much more efficient—in circuit models, and in reality—when circuits are instead designed to support many parallel array accesses. In particular, two-dimensional circuit-layout models support two-dimensional sorting networks such as [117] or [110]: circuits of mass $n^{1+o(1)}$ that sort n integers, each integer having $n^{o(1)}$ bits, in time $n^{1/2+o(1)}$, for price-performance ratio $n^{3/2+o(1)}$. The real-world scalability of these circuits is demonstrated by, e.g., the FPGA implementation in [60]. Presumably large-scale attackers would use ASICs rather than FPGAs.

Three-dimensional models and circuits improve $n^{3/2+o(1)}$ to $n^{4/3+o(1)}$, although it is far less clear that this can be physically realized. For the Boolean-circuit model in Section 2.1, the same sorting task costs $n^{1+o(1)}$. Bit-operation counts are a lower bound on price-performance ratio, not an upper bound; the gap between 1 and 3/2 comes from the communication costs reviewed in Appendix F.6.

For comparison, the RAM cost of sorting depends on the choice of a RAM model and of a cost metric (the same way that different choices produce variations in Appendix D), but even a heavily restricted RAM model would allow radix sort of n integers, each having b bits, to finish in "time" $O(bn)$. For comparison, the sorting circuits inside CAT use about $(1/4)bn(\log_2 n)^2$ bit operations, and the best asymptotic results known are $\Theta(bn \log n)$ with a much larger Θ constant. These gaps show that RAM metrics, despite having the same n exponent, are considerably below bit-operation counts for sorting—and thus considerably farther from price-performance ratio than bit-operation counts are.

To summarize, moving from circuit models to RAM models would not just complicate definitions but also move farther from reality, as illustrated by basic subroutines such as random access and sorting. This is not saying that circuit models are perfectly realistic (see Appendix F.6); it is saying that RAM models are worse.

G.4. Do RAM metrics prevent hidden precomputation? A standard criticism of circuit models in computational-complexity theory is that one can build a family $(C_0, C_1, \dots)$ of polynomial-size circuits that computes a function that cannot be computed in polynomial time—even an uncomputable function, such as the halting function. Specifically, take any uncomputable function $n \mapsto f(n)$ from $\{0, 1, 2, \dots\}$ to $\{0, 1\}$, and define circuit C_n as the circuit that maps all n-bit strings to $f(n)$.

The standard fix for this problem is to require another algorithm P that, given n, precomputes the circuit C_n. Syntactically, this structure does not allow P to be a circuit, since P needs to allow an arbitrarily large integer n as input; P is instead defined as, e.g., a Turing machine, or perhaps an algorithm in some RAM model.

For example, if this algorithm P is required to run in time bounded by a polynomial in n, then its output C_n necessarily has size bounded by a polynomial in n, and the overall function computed by the circuit family $(C_0, C_1, \dots)$ is a polynomial-time function. More generally, one can impose a cost limit on P as a metric for the "uniformity" of the family of circuits, and impose another cost limit on each C_n.

This background creates a perception that circuit models are blind to any amount of precomputation—perhaps much more than the circuit cost, perhaps not even computable—and that "uniform" models, including RAM models, have the advantage of seeing precomputation.

However, as illustrated by the attacks in [27], this advantage disintegrates when the evaluation of algorithm costs is limited to any finite range of n, which is the situation in real-world cryptography and in real-world algorithm experiments. What follows is a concrete example.

Consider the elliptic-curve discrete-logarithm attacks from [27], algorithms A_n that compute n-bit discrete logarithms in RAM "time" $(2 + o(1))^{n/3}$, far below the conventional $(2 + o(1))^{n/2}$. As emphasized in [27], these attacks do not appear to be a real-world threat to deployed systems with $n = 256$: the only published algorithm P that maps n to A_n takes much more "time", namely $(2 + o(1))^{2n/3}$.

Consider the following attempt to formalize the apparent difficulty of finding A_n: build a framework that measures the cost of program P, in this case $(2 + o(1))^{2n/3}$, and test this by checking various small values of n.

In response, someone secretly computes A_n for all small n, and builds a new program P' that simply includes A_n for those n, while falling back to the same behavior as P for larger n. The framework will then measure P' as being very fast for every n that it tries. The framework has been blinded to the precomputation, in the same way as a framework that simply considers A_n from the outset. As an informal countermeasure, one might inspect P' to check whether something interesting is happening for small n, but the same countermeasure is equally applicable to CAT, so this does not show an advantage of RAM models.

The uniformity notions typically considered in computational-complexity theory, such as whether P takes time polynomial in n, are not vulnerable to this type of replacement:

- Including precomputations for any finite range of n has no effect on whether P takes time polynomial in n.
- Including precomputations for infinitely many n is not compatible with the basic requirement of P being a finite-length program.

This protection is an artifact of the purely asymptotic definitions, where a finite-length program handles infinitely many n and where strange behavior for any particular n is disregarded. This is of no help in formally defining attack costs for, e.g., $n = 256$.

G.5. Do RAM metrics improve optimization quality? Finally, an interesting argument for RAM models is the idea that, even though assigning cost just 1 for access to arbitrarily large arrays is unrealistic, it is also simple and familiar, improving the chance that algorithms will be successfully optimized for such cost metrics. See, e.g., the statement quoted in Appendix D.5. However, in the Kyber-512 and AES-128 examples in Appendix D, the literature (1) explicitly allowed random access as a cost-1 "gate", (2) missed very easy speedups exploiting this, and (3) was more successful at optimizing bit operations when this random-access "gate" was ignored.

H Limitations in CAT

This paper focuses on a non-quantum model of computation. The literature also studies quantum ISD algorithms, replacing the combinatorial searches in ISD with Grover's algorithm and, more generally, replacing the random walks in ISD with quantum walks. Efficiently simulating formalizations of these algorithms, when the simulator does *not* have a quantum computer, would require a formalized framework to simulate quantum walks. This paper's simulator does not have any specific knowledge of random walks; the random walks are encapsulated inside constructions of ISD algorithms and derivations of cost/probability formulas.

The way that `circuitprob` tries an attack circuit C, namely generating a pair (P, s) and checking whether $C(P) = s$, captures many problems of interest in cryptography (e.g., OW-CPA problems for PKEs and the AES-128 key-search problem in Appendix I) but certainly not all. Allowing a more complicated comparison function between $C(P)$ and s would support PRG problems (as explained in [87, Section 3]) and various multi-target problems. Allowing C to call oracles would support PRF problems.

CAT's simulator is also not integrated with any particular proof system. The user has to check manually that, e.g., the OW-CPA problem used here matches the OW-CPA problem hypothesized to be secure inside a proof of a more complicated security property. Analogously, in the case of an algorithm analysis that is rigorously proven modulo a conjecture, the simulator might be usable for checking the conjecture, but the user would have to manually check that what the simulator sees matches the hypothesis of the proof. Integration would reduce the risks of interface mismatches.

An intrinsic limitation of step-by-step simulations, with or without this paper's formalization, is that their cost grows at least linearly with the number of steps (and often even more than linearly when increased memory usage triggers increased memory-access costs on the CPU carrying out the simulation). One *hopes* that any prediction errors that appear for cryptographic sizes will already appear in scaled-down simulations having much lower cost, but there is no guarantee of this. As Section 1.1 explains, this paper is starting from attack analyses in the literature that are not proofs; formalizing these analyses reduces the risk of error, but it does not produce a formally verified proof.

As noted in Section 3, adding malicious attack code to CAT can sabotage CAT's results. Human review can help but will not necessarily catch, e.g., an attacker tweaking probability formulas in a way that does not show up in small-scale simulations.

I AES-128 in CAT

CAT includes formalizations of (1) an `aes128` problem and (2) an `aes128_enum` brute-force attack against this problem. This appendix describes the problem, the attack, and various cost reductions not included in the attack. This example illustrates that CAT is not limited to ISD.

This attack searches the *complete* AES-128 key space using fewer than $2^{142.89}$ bit operations. To compare the costs of AES-128 key search to the costs of ISD, one should instead consider the cost of finding an *average* AES-128 key, as noted in Appendix D.4; this cost is below $2^{141.89}$ bit operations with this attack.

One reason for interest specifically in the cost of a brute-force attack against AES-128 is that NIST

- selected this cost as the minimum security level allowed in the NIST Post-Quantum Cryptography Standardization Project and
- made decisions in that project on the basis of very close comparisons to this cost.

See Appendix A.3. For definitional issues with NIST's previous comparisons, see Appendix D.

I.1. AES-128 key search. The `aes128` problem in CAT has two parameters: integers K and C with $1 \le K \le 128$ and $1 \le C \le 128$. The secret information is a K-bit string s. An AES-128 key derived from s is used to encrypt two public plaintext blocks. The first C bits from each ciphertext block are also public.

Specifically, define k as follows: zero-pad s to 128 bits and then view the result as a 16-byte AES-128 key k. The public information is (p_0, c_0, p_1, c_1), where p_0 and p_1 are 16-byte strings, c_0 is the first C bits of $\mathrm{AES}_k(p_0)$, and c_1 is the first C bits of $\mathrm{AES}_k(p_1)$. The strings s, p_0, p_1 are chosen independently and uniformly at random. Bits inside bytes are viewed in little-endian order, although other orderings would be compatible with the comments below.

Note that the case $K = 128$ generates the 16-byte secret key k uniformly at random; this matches typical uses of AES-128. Decreasing K is dangerous for security: it forces $128 - K$ bits of k at specified positions to be 0, making k easier to guess by a factor 2^{128-K}. However, this generalization is important for testability of brute-force attacks, the same way that scaling down problem parameters in Section 3.5 is important for testability of ISD algorithms.

The K secret bits in k produce $2C$ bits in (c_0, c_1). For example, in the case $C = 128$, all 256 bits of $\mathrm{AES}_k(p_0)$ and $\mathrm{AES}_k(p_1)$ are public. Presumably this almost always[7] determines k. As $2C$ decreases down towards K, presumably the probability of key collisions increases; Appendix K describes CAT's model of this probability.

I.2. Enumerating AES-128 keys. The `aes128_enum` attack in CAT has four parameters: an integer I between 0 and $2^K - 1$; an integer `QX`, either 0 or 1; an integer `QU`, at least 1; and an integer `PE`, at least 1. The `PE` parameter is actually computed as `QU` times another parameter `QF`.

[7] Removing the word "almost" would not make a reasonable conjecture. The case $p_0 = p_1$ occurs with probability 2^{-128}, and it would be surprising for the map from k to $\mathrm{AES}_k(p_0)$ to be injective. One could add testability for this gap by introducing a P parameter for the number of nonzero bits in p_0 and p_1, analogous to the K parameter for k. It would also be useful to test attacks against generalizations of AES that scale 16 8-bit bytes down to 16 b-bit words for $b \ge 1$.

The attack carries out I iterations. Each iteration considers one guess g for the K-bit secret. The first iteration guesses an all-0 string, and each subsequent iteration guesses the next string in reverse lexicographic order.

If QX is 0 then each iteration encrypts plaintexts p_0 and p_1 under g, comparing the outputs to c_0 and c_1 respectively. The QU and PE parameters are ignored in this case.

If QX is 1 then each iteration encrypts p_0 under g, compares the output to c_0, and, if there is a match, inserts g into a queue of size QU, which is checked and cleared every PE iterations. The check encrypts p_1 under g and compares the output to c_1, for each g in the queue.

With either choice of QX, the attack returns the all-1 string by default if it does not detect any g as mapping p_0 and p_1 to c_0 and c_1 respectively.

The advantage of taking QX to be 1, specifically with PE larger than QU, is that PE iterations try encrypting p_1 only QU times rather than PE times. Disadvantages are the costs of queue management and the risk of the correct key being pushed out of the queue before it is checked. The analysis accounts for these effects, and concludes that taking QX to be 1 saves a factor $2^{0.98}$ overall.

At a lower level, this attack computes the AES S-box using the 113-bit-operation circuit from [94], credited in [94] to Calik as an improvement over the 115-bit-operation circuit from [35]. This attack follows the original definition of AES-128 to straightforwardly convert the S-box into a full encryption circuit.

I.3. The analysis. The `aes128_enum_cost` function straightforwardly tracks the steps in `aes128_enum`. The `aes128_enum_prob` function starts with $I/2^K$, the chance that one of the I guesses matches the secret; accounts for queue losses in the case QX = 1, using the model from Appendix J; and then adds $1/2^K$ to account for the chance that the default all-1 string matches the secret. CAT also automatically accounts for multiple preimages as explained in Appendix K.

CAT includes an `aes128.py` script that uses `searchparams` to find attack parameters for $K = C = 1$, then $K = C = 2$, and so on through $K = C = 128$, in each case choosing $I = 2^K - 1$ to focus on attacks that enumerate all $2^K - 1$ non-default keys. CAT also includes an `aes128-table.py` script that converts the `aes128.py` output into Table 2. The scripts include $C = K - 1$ for $K \in \{2, 3, 4\}$ to illustrate the effects of varying K and C independently.

The parameters found for $K = C = 128$ are QX = 1, QU = 1, and PE = 2048, with predicted cost approximately $2^{142.882195}$ and predicted success probability approximately 1. Amortizing the cost of encrypting p_1 across 2048 iterations makes the cost almost unnoticeable. It is, furthermore, intuitively clear that whichever 2048 iterations find the secret would have to be extraordinarily unlucky to make another guess that matches the same 128 ciphertext bits, overflowing a size-1 queue and losing the secret. Note that this is not a proof.

The `aes128.py` script also runs `circuitcost` and `circuitprob` (with `trialfactor` = 100000) on the parameters found for $K \le 10$. This detects a statistically significant discrepancy for $K = C = 2$, where parameters QU = 1 and PE = 4 succeed 2.78% more often than predicted; perhaps refinements of the model in Appendix K can explain this discrepancy. The discrepancy disappears as K and C increase within the range covered by `circuitprob`. This does not rule out risks of mispredictions for larger K, analogous to the ISD risks covered in Appendix L.

Table 2: Predicted performance of various parameters for `aes128_enum` with $I = 2^K - 1$, and observed performance for $K \le 10$. The QX, QU, and QF columns are the attack parameters selected by `searchparams`. The "lgcost" column is the logarithm base 2 of the predicted attack cost. The "prob" and "prob2" columns are the predicted success probability, respectively without and with the collision handling from Appendix K. The "succ" column is the observed success probability in 100000 trials. The last four columns are rounded to the number of digits displayed after the decimal point.

K	C	QX	QU	QF	lgcost	prob	prob2	succ
1	1	0	1	1	15.838145	1.00000	0.87500	0.87571
2	1	1	2	2	17.160679	0.93750	0.66000	0.67227
2	2	1	1	4	16.838650	0.82812	0.76991	0.79131
3	2	1	3	4	18.161540	0.95758	0.78013	0.79134
3	3	1	2	4	18.009326	0.94691	0.89966	0.90680
4	3	1	4	4	19.088768	0.98028	0.87545	0.88063
4	4	1	3	8	19.010400	0.98619	0.95823	0.96095
5	5	1	3	16	19.928685	0.98174	0.96730	0.96886
6	6	1	4	16	20.908913	0.99639	0.98879	0.98873
7	7	1	4	48	21.877120	0.99603	0.99219	0.99212
8	8	1	4	64	22.861344	0.99584	0.99392	0.99411
9	9	1	4	128	23.853780	0.99575	0.99478	0.99500
10	10	1	5	256	24.853332	0.99932	0.99883	0.99883
11	11	1	4	256	25.851858	0.99963		
12	12	1	3	384	26.849737	0.99932		
13	13	1	2	256	27.849946	0.99937		
14	14	1	2	512	28.847720	0.99937		
15	15	1	2	512	29.848254	0.99984		
16	16	1	2	512	30.848765	0.99996		
32	32	1	1	2048	46.849717	1.00000		
48	48	1	1	2048	62.855181	1.00000		
64	64	1	1	2048	78.860624	1.00000		
80	80	1	1	2048	94.866047	1.00000		
96	96	1	1	2048	110.871450	1.00000		
112	112	1	1	2048	126.876832	1.00000		
128	128	1	1	2048	142.882195	1.00000		

Running `circuitcost problem=aes128 K=1` and so on through `circuitcost problem=aes128 K=8` considers many different attack parameters and finds `aes128_enum_cost` correctly predicting costs in all cases. Running `circuitprob` for various parameters finds discrepancies above 2.78%: for example, `circuitprob` with $K = 4$, $C = 1$, $I = 8$, QX = 1, QU = 1, QF = 4, `trialfactor` = 100000 observes success probability approximately 0.197, while the prediction is approximately 0.186 accounting for multiple preimages (and 0.296875 otherwise).

Presumably the accuracy of `aes128_enum_cost` could be formally verified, but `aes128_enum_prob` is a different matter: proving reasonably tight lower bounds on success probability is an open

problem. It is remarkable that the general difficulty of proving effectiveness of state-of-the-art attacks (see Appendix B) is visible even for an attack as simple as brute-force search for a cipher key.

I.4. Cost reductions not included in this attack. The cost of approximately $2^{142.882195}$, i.e., $2^{14.882195} \approx 30198.6$ per iteration, consists of the following components:

- Cost 14.6 per iteration for the handling of p_1 every 2048 iterations. Increasing `PE` beyond 2048 would decrease this cost; `searchparams` skips parameter modifications that produce only tiny improvements.
- Cost 256 per iteration to compare to 128 ciphertext bits. Constant folding would decrease this to 255. Limiting the comparison to, e.g., 20 ciphertext bits would decrease this to 39, at the expense of re-encrypting p_0 along with each encryption of p_1.
- Cost 256 per iteration to move to the next key guess. This could be almost entirely eliminated with unrolling, Gray codes, etc.
- Cost 386 per iteration to conditionally insert the current 128-bit guess into the queue. NOT folding would save 1 operation. Some bits could simply be skipped at the expense of a brute-force search along with each encryption of p_1, although this seems less beneficial than decreasing the number of ciphertext bits compared.
- Cost 29286 per iteration to encrypt p_0.

The following comments focus on ways that cost could be reduced inside the encryption of p_0.

Cost 5910 per iteration is spent on AES key expansion, which can trivially be precomputed, reducing overall costs below $2^{142.568}$ (and $2^{141.568}$ on average). The storage of $2^{128}-1$ expanded keys would be problematic in metrics that account for circuit mass, but the generic observation that adjacent keys share portions of computations already produces some benefit with much less storage; perhaps the decomposition of [81] is also applicable here. Part of the circuit to encrypt p_0 is also shared across adjacent keys.

Note that if p_0 were constant then the more sophisticated precomputation of [65] would provide better tradeoffs between storage and main computation. This is the situation in many applications, but not in the uniform-random-p_0 problem formalized in CAT.

If the initial comparison to c_0 is limited to, e.g., 32 bits—at the expense of occasionally re-encrypting p_0, as noted above—then, for reasonable choices of bit positions, eliminating unused operations automatically eliminates many of the final computations inside encryption. For more sophisticated speedups along these lines, see [33], [32], and [116].

For simplicity, this attack uses bit operations to compute and apply the AES round constants. Constant folding would save most of these operations, but this is a negligible cost in any case.

J Accounting for queue losses and window losses

The high-level description of `isd1` in Section 4.8 allows the set $S^{(1)}$ to be smaller than the set of all collisions. The circuits in CAT exploit this flexibility in two ways described in Section 5.11:

- The circuits check for colliding elements in a sorted list L only at positions having distance at most `WI`. `WI` stands for "window". Collisions at larger distances are lost.
- Instead of having further processing (namely, computing $s^{(1)} - (T_L^{(1)}[I] + T_R^{(1)}[I])$ and testing its Hamming weight) along with every collision test, the circuits push collisions into a queue for further processing later. The queue can hold at most `QU` collisions. The queue is periodically processed and cleared, but if it overflows in the meantime then some collisions are lost.

Similar comments apply to both levels of collision search in `isd2`. For `isd0`, there is no collision search, but there is still a queue, except when $\ell = 0$. There is also a queue for `aes128_enum`; see Appendix I.

This appendix explains how CAT predicts the probability that an iteration will miss a solution because the solution is pushed out of a fixed-size queue or because it is beyond a fixed-size window.

One could use these predictions to take window lengths and queue lengths large enough that there is negligible probability of a solution being missed; `searchparams` instead aims for the best ratio between total circuit cost and success probability.

J.1. Queue analysis. Consider P circuits producing events. Model the events as a Bernoulli process: producer 0 generates an event with probability ϵ, producer 1 generates an event with probability ϵ, etc., each of these events being independent.

Now consider P circuits consuming whichever events were produced. This is wasteful if ϵ is small: the average number of events produced is only ϵP, so most of the consumer circuits are useless at any moment.

So instead consider P producer circuits pushing their results into a queue of length $Q \le P$, followed by Q consumer circuits processing the results in the queue. If more than Q events are produced then only Q events will be consumed; the remaining events will overflow the queue and will be lost. The following paragraph quantifies this loss.

Define $\varphi \in \mathbb{R}[x]$ as the polynomial $1 - \epsilon + \epsilon x$. The probability that the P producer circuits produce exactly e events is φ_e^P, meaning the coefficient of x^e in the polynomial φ^P. The average number of events consumed is thus $\sum_{e<Q} e\varphi_e^P + \sum_{e\ge Q} Q\varphi_e^P$. This is the same as $Q - \sum_{e<Q}(Q-e)\varphi_e^P$ since $\sum_e \varphi_e^P = \varphi^P(1) = 1$.

In other words, the average number of events produced is ϵ per producer, but the average number of events consumed is only $(Q - \sum_{e<Q}(Q-e)\varphi_e^P)/P$ per producer. This formula uses Q coefficients $\varphi_0^P, \ldots, \varphi_{Q-1}^P$ of φ^P.

J.2. Window analysis. Consider the following general collision-finding scenario. There are two lists. The first list contains $A > 0$ pairs (s,t). The second list contains $B > 0$ pairs (s',t'). Sort all $(s,0,t)$ in lexicographic order together with all $(s',1,t')$, and check all pairs of positions in the sorted list with distance at most w to see whether the list entries at those positions have the form $(s,0,t)$ and $(s',1,t')$ with $s = s'$.

Model each s and each s' as an independent uniform random element of $\mathbb{F}_2^\ell$. Define $\psi \in \mathbb{R}[x]$ as the polynomial $1 - 1/2^\ell + x/2^\ell$. For any particular $s \in \mathbb{F}_2^\ell$, the chance that s appears exactly e times

in the first list is ψ_e^A, and the chance that s appears exactly f times in the second list is ψ_f^B.

If s appears (e, f) times then there are exactly ef collisions involving s. However, only positions having distance at most w are checked, and this loses some collisions if $e + f > w + 1$.

More precisely, the rightmost $(s, 0, t)$ finds $\min\{f, w\}$ collisions; the previous $(s, 0, t)$ finds $\min\{f, \max\{w - 1, 0\}\}$ collisions; and so on through the first $(s, 0, t)$, which finds $\min\{f, \max\{w - e + 1, 0\}\}$ collisions. In other words, in an $e \times f$ array of dots, one counts the number of dots on the first w diagonals. This is

$$C_w(e, f) = \begin{cases} w(w+1)/2 & \text{if } w \le m, \\ m(m+1)/2 + m(w-m) & \text{if } m < w \le M, \\ ef - (e+f-w)(e+f-w-1)/2 & \text{if } M < w \le e+f, \\ ef & \text{if } e+f < w \end{cases}$$

where $m = \min\{e, f\}$ and $M = \max\{e, f\}$.

To summarize, there is probability $\psi_e^A \psi_f^B$ that s appears exactly e times in the first list and exactly f times in the second list, and this event gives $C_w(e, f)$ collisions involving s. The average number of collisions involving s is $\sum_{e,f} \psi_e^A \psi_f^B C_w(e, f)$.

Now let s vary, and sum over all s: the average number of collisions in total is $2^\ell \sum_{e,f} \psi_e^A \psi_f^B C_w(e, f)$.

As in the queue analysis, one can rewrite this sum as a sum with fewer terms. Specifically, abbreviate $\sum_{e \ge w} \psi_e^A$ as S and $\sum_{f \ge w} \psi_f^B$ as T. Then $S = 1 - \sum_{e<w} \psi_e^A$ and $T = 1 - \sum_{f<w} \psi_f^B$, since $\sum_e \psi_e^A = 1$ and $\sum_f \psi_f^B = 1$. Now split (e, f) into the following four regions:

- $e \ge w$ and $f \ge w$: One has $C_w(e, f) = w(w+1)/2$, which is independent of (e, f), so $\sum_{e \ge w, f \ge w} \psi_e^A \psi_f^B C_w(e, f) = (w(w+1)/2)ST$.
- $e < w$ and $f \ge w$: One has $C_w(e, f) = e(e+1)/2 + e(w-e)$, which is independent of f, so $\sum_{e<w, f \ge w} \psi_e^A \psi_f^B C_w(e, f) = \sum_{e<w} \psi_e^A C_w(e, f) T$.
- $e \ge w$ and $f < w$: One has $C_w(e, f) = f(f+1)/2 + f(w-f)$, which is independent of e, so $\sum_{e \ge w, f<w} \psi_e^A \psi_f^B C_w(e, f) = \sum_{f<w} \psi_f^B C_w(e, f) S$.
- $e < w$ and $f < w$: There are only $(w-1)^2$ terms in this region (with nonzero e, f).

This reduces the computation of the average number of collisions to a sum of approximately w^2 terms.

J.3. Windows into queues. Inside `isd1`, pairs of list entries at positions separated by at most `WI` are checked *in a random order* for whether they are collisions, and pushed into a queue of length `QU` for further processing, where the processing occurs after every `PE` checks. As in Appendix I, the `PE` parameter is actually computed as `QU` times another parameter `QF`. Manual parameter optimization would take `QF` somewhat below the reciprocal of the queue-push probability.

Given w = `WI` and the original list sizes, CAT uses the formulas from Appendix J.2 to predict the average number of collisions found, under the heuristic that the relevant elements of $\mathbb{F}_2^\ell$ are sufficiently random. Then CAT heuristically treats the queue insertions as a Bernoulli process, with probability determined by the collision prediction, and applies the formulas from Appendix J.1, with Q = `QU` and P = `PE`, to predict the average number of collisions consumed from the queue.

(Note that checking for collisions at pairs of list positions in lexicographic order, rather than a random order, would break the Bernoulli-process model: any s that appears several times would produce a burst of consecutive events, probably overloading a short queue and certainly not matching the independence assumption.)

Similar comments apply to the two levels of collision search in `isd2`. The intermediate list sizes here are variables, but heuristically have a narrower and narrower distribution around their predicted sizes as parameters increase. The predicted sizes are averages that are not necessarily integers; the formulas in Appendix J.2 can be applied to non-integral A, B.

For reliable computations on real numbers, CAT uses the existing MPFI [104] library for interval arithmetic, repeatedly doubling precision (starting with 32 bits) until the final probability-prediction intervals have relative width below 2^{-20}.

K Accounting for multiple preimages

Fix finite sets X and Y. Fix a function $F : X \to Y$. Consider a secret e chosen uniformly at random from X, and consider the problem of finding e given $y = F(e)$.

The set $F^{-1}(y)$ of preimages of y could be larger than $\{e\}$. Then e is information-theoretically hidden among the preimages. This limits the success probability of any algorithm that outputs an element of X as a guess for e: for example, if there are two preimages, then the success probability is at most 50%. An algorithm is required to output an element of X in, e.g., the OW-CPA security definition, or in the context of "unique decoding" in coding theory.

Typical analyses of ISD algorithms assume a different interface, a "list decoding" interface in which an algorithm outputs a list of elements of X and succeeds if e is in the list. In other words, the algorithm succeeds if it encounters the desired e; if it also encounters another preimage x then it can output both e and x, without having to make a choice between e and x. The algorithm analysis then simply asks whether e is encountered, without worrying about whether other preimages also appear.

K.1. Do multiple preimages matter? One can convert a list-decoding algorithm to a unique-decoding algorithm, or more generally convert a list-of-preimages attack into an OW-CPA attack, by simply printing the first preimage of y in the list, or printing a uniform random element of X if there is no preimage in the list.

The only way for this conversion to reduce success probability is for there to be multiple preimages of y. There are two easy ways to argue that ISD algorithms are unlikely to encounter multiple preimages:

- In the context of applying ISD to attacking the OW-CPA property of the McEliece cryptosystem: The map from plaintexts to ciphertexts is injective—this follows immediately from, e.g., the fact that the McEliece decryption algorithm always works—so there is never more than one preimage.
- In the context of applying ISD to decoding for a uniform random matrix: Write H for the full parity-check matrix obtained by gluing the uniform random matrix to an identity matrix. The input e is a weight-t element of $\mathbb{F}_2^n$; the output

is a syndrome $y = Hx \in \mathbb{F}_2^{n-k}$. If a weight-$t$ vector $x \neq e$ has $y = Hx$ then $H(x-e) = 0$, which has probability 0 if $x - e$ is supported on the identity part of H and probability $1/2^{n-k}$ otherwise. The total chance of another preimage is thus at most $(\binom{n-k}{t} - 1)/2^{n-k}$, which is negligible for parameters (n, k, t) of cryptographic interest.

However, the effect of multiple preimages is easily visible in ISD for a uniform random matrix with small (n, k, t). For example, for $(n, k, t) = (16, 12, 1)$, straightforward computer experiments show that a complete search through preimages succeeds with probability only about 65.4%.

Small parameters are important for this paper. Small parameters make it feasible to experimentally evaluate success probabilities for comparison to predicted success probabilities. One can dismiss the discrepancy between 100% and 65.4% as not being very large, but ignoring such discrepancies can easily hide other discrepancies that do *not* disappear as sizes increase.

K.2. Options for addressing the discrepancy. One way to eliminate multiple preimages would be to run attack experiments against ciphertexts for parity-check matrices for Goppa codes, as in the McEliece cryptosystem. For example, the key-generation software from [8] takes only 180 million cycles on an Intel Skylake core for `mceliece6960119f`, which is designed for long-term security. One can save more time by using a single key for attack experiments with many ciphertexts.

On the other hand, that key-generation software is not designed to support parameters smaller than cryptographic sizes, and writing new software for carrying out experiments would raise verification questions. There is value in the simplicity of considering uniform random matrices as an attack target, as in the ISD literature—but precise analyses then require accounting for multiple preimages.

A different way to eliminate multiple preimages would be to consider the problem of recovering e from, say, $F(e)$ and a cryptographic hash of e; often attackers are facing problems of this type. Another x with $F(x) = F(e)$ would be very unlikely to have the same hash as e. Buffering several preimages, and occasionally computing hashes to exclude preimages with the wrong hash, would have very low cost, and the buffer would almost never overflow.

This appendix takes another approach: quantifying the impact of multiple preimages. The uniform-random-function model below is broadly applicable and already captures most of the impact, although a closer look at ISD obtains more accurate results.

K.3. The uniform-random-function model. Consider a search through a nonempty subset S of X. Perhaps this is a brute-force search, or perhaps something faster; the speed does not matter for the following analysis. Assume that the search outputs one of the preimages it finds, and aborts if it does not find any preimages.

Model F as a uniform random function from X to Y. Any particular $x \in X - \{e\}$ then has $F(x) = F(e)$ with probability $1/\#Y$. In other words, for each $i \in \{0, 1\}$, the probability that x contributes i additional preimages of $F(e)$ is the coefficient of z^i in φ, where $\varphi \in \mathbb{R}[z]$ is the polynomial $1 - 1/\#Y + z/\#Y$.

The search succeeds with chance $(\#S/\#X) \sum_{i \geq 0} \varphi_i^{\#S-1}/(i+1)$, where (as in Appendix J) $\varphi_i^{\#S-1}$ means the coefficient of z^i in the polynomial $\varphi^{\#S-1}$:

- $\#S/\#X$ is the chance that $e \in S$. A conventional analysis would stop at this point, saying that the search finds e with probability $\#S/\#X$.
- Given that $e \in S$, there are $\#S - 1$ elements $x \in S - \{e\}$, each x having an independent probability of $F(x) = F(e)$; so the chance of $\#\{x \in S - \{e\} : F(x) = F(e)\} = i$ is $\varphi_i^{\#S-1}$ for each $i \in \{0, 1, 2, \ldots\}$.
- Given that $e \in S$ and that $\#\{x \in S - \{e\} : F(x) = F(e)\} = i$, the search succeeds with probability $1/(i+1)$.

One can use the formula $\varphi_i^{\#S-1} = \binom{\#S-1}{i}(1 - 1/\#Y)^{\#S-1-i}(1/\#Y)^i$ to compute $\sum_{i \geq 0} \varphi_i^{\#S-1}/(i+1)$, but it is easier to instead note that $\sum_{i \geq 0} \varphi_i^{\#S-1}/(i+1) = \int_0^1 \varphi^{\#S-1}\, dz = (\#Y/\#S)(1 - (1 - 1/\#Y)^{\#S})$. The search thus succeeds with chance $(\#Y/\#X)(1 - (1 - 1/\#Y)^{\#S})$.

Here are two numerical examples:

- $\#X = 16$, $\#Y = 16$, and $\#S = 16$. The success probability in this model is then $1 - (1 - 1/16)^{16}$, about 64.4%, slightly below the actual 65.4% chance mentioned above. The conventional approximation is 100%.
- $\#X = 16$, $\#Y = 16$, and $\#S = 4$. The success probability in this model is then $1 - (1 - 1/16)^4$, about 22.8%. The conventional approximation is 25%.

For larger parameters, in particular with $\#S/\#Y$ converging to 0, the ratio between the success chance $(\#Y/\#X)(1 - (1 - 1/\#Y)^{\#S})$ in this model and the conventional $\#S/\#X$ converges to 1, matching the intuition that multiple preimages become less and less common.

The prediction mentioned in Section 6.1, labeled `prob2` inside CAT, is computed via this model as follows: first a simplified prediction, labeled `prob`, is computed without accounting for collisions; then `prob2` is computed as $(\#Y/\#X)(1 - (1 - 1/\#Y)^{\#X \cdot \texttt{prob}})$. Section 6.2 uses only `prob`.

K.4. Accounting for local injectivity. Consider any matrix $H \in \mathbb{F}_2^{(n-k) \times n}$ of the form $(T|I)$, where T is an $(n-k) \times k$ matrix and I is the $(n-k) \times (n-k)$ identity matrix. Consider the following search for a weight-t vector $e \in \mathbb{F}_2^n$ given $He \in \mathbb{F}_2^{n-k}$: if He has weight t then output k zeros followed by He. This is an example of one iteration of Prange's original ISD algorithm.

This search finds e if and only if $e \in S$, where S is the set of weight-t elements of $\mathbb{F}_2^n$ starting with k zeros. The search cannot encounter another preimage at the same time: H is injective on this set S. Note that $\#S = \binom{n-k}{t}$.

For example, for $(n, k, t) = (16, 12, 1)$, this search succeeds for exactly $\binom{4}{1} = 4$ choices of e. The phenomenon of multiple preimages does not occur here: the success probability for one iteration of Prange's algorithm is exactly the conventional 25%, not the 22.8% from the uniform-random-function model.

On the other hand, the success probability of many iterations of Prange's algorithm for a uniform random T is certainly not the conventional 100%: it cannot exceed the 65.4% mentioned above.

In the formula $(\#S/\#X) \sum_{i \geq 0} \varphi_i^{\#S-1}/(i+1)$ from the uniform-random-function model, $\#S - 1$ counts the number of elements of S that have a chance of colliding with e under F, given that $e \in S$. In Prange's algorithm, the elements of S excluded from collisions are not just e itself, but another $C - 1$ elements of S, where $C = \binom{n-k}{t}$.

This suggests that

$$\frac{\#S}{\#X}\sum_{i\geq 0}\frac{\varphi_i^{\#S-C}}{i+1}=\frac{\#S}{\#X}\frac{\#Y}{\#S-C+1}\left(1-\left(1-\frac{1}{\#Y}\right)^{\#S-C+1}\right)$$

would be a better approximation to the success probability.

In improved versions of Prange's algorithm with restricted column swaps, there are more elements of S visibly excluded from collisions. Consider, for example, two iterations, where the kth column is swapped with the last column between the iterations. Then S is the set of weight-t elements of $\mathbb{F}_2^n$ that start with k zeros or that start with $k-1$ zeros and end with one zero. If e starts with k zeros and ends with one zero then it cannot collide with any other elements of S under H.

K.5. Accounting for excluded information sets. If H has the form $(0|I)$ then there is only one information set for H, namely the set containing the first k positions. Prange's algorithm always uses this information set, and finds e only if e is 0 on the first k positions.

This is related to the fact that there are many collisions under H, but the consequences are more extreme. The problem is not merely that the algorithm has to guess from among many preimages; the problem is that, after the first iteration, subsequent iterations of the algorithm simply repeat searching the same space.

More generally, if e has a bit set at a position where H has a 0 column, then Prange's algorithm will never find e. As a numerical example, this occurs with probability 3/64, i.e. 4.6875%, in the case $(n,k,t)=(16,12,1)$: there is chance 3/4 that e is in the random part of H, and then chance 1/16 that H has a 0 column at that position. Experiments show that Prange's algorithm succeeds with probability 62.1%.

L Risks of mispredictions

This appendix considers ways that inaccuracies could have appeared in the predictions in Section 6.2 while avoiding detection by the simulations in Section 6.1; and, more broadly, ways that Table 1 can deviate from the actual cost of attacks.

Figure 4 shows a very close match—always within 0.2 bits, usually even closer—between cost/probability predictions and actual circuit behavior, as shown by circuit simulations, for parameters obtained by `searchparams`, across a range of three doublings of n. Consider the hypothesis that this remains true for six more doublings, covering sizes proposed for use in cryptography: in particular, that the predictions calculated in Section 6.2 match the actual circuit behavior within 0.2 bits.

There is an obvious way that this hypothesis could fail: the predictions for an attack could have an inaccuracy growing with the problem parameters. The graph seems to show increasing accuracy with problem parameters, but this could be because the predictions have some inaccuracies visible only at small sizes and other inaccuracies visible only at large sizes, with the right side of the graph between these sizes.

One way to try to catch such inaccuracies is to carry out larger experiments. Another way, possibly more efficient, is to carry out more experiments for small sizes, checking for very small discrepancies between predictions and simulations, with the hope that an inaccuracy for large sizes will appear as a detectable inaccuracy for small sizes. This requires resolving all issues that appear for small sizes, even when it is clear that those issues disappear for larger sizes; the handling of collisions in Appendix K is an example of a step towards this.

If the `searchparams` choice of an attack parameter happens not to vary throughout the range of problem parameters considered in simulations, then a formula for the impact of that attack parameter would be checked only for that particular choice. Experience shows that human errors in generating formulas are frequently caught by single tests, but presumably further tests make errors less likely. This is why Section 6.1 specifically moves the pair (`IT`,`RE`) from $(1,1)$ to $(2,1)$ to see the cost and probability impact of resets, while moving from $(1,1)$ to $(4,4)$ to see the impact of random walks. However, this does not enforce variations in all parameters, and it is in any case possible that an error in formulas escapes detection for whichever parameters are tried.

Another risk is as follows. Assume that a particularly effective portion of the parameter space for an attack is predicted to be much worse, because of a prediction error that applies only to that portion of the parameter space. Presumably `searchparams` will avoid those parameters, so tests of parameters selected by `searchparams` will not catch the prediction error. This would not contradict the hypothesis stated above, but it would mean that prediction errors are limiting the effectiveness of the circuits considered. To the extent that cryptanalysis papers include experiments, they typically carry out experiments only for "optimized" parameters, incurring the same risk.

One way to address this risk at moderate cost would be to carry out simulations for parameters *considered* by `searchparams` rather than just parameters *selected* by `searchparams`. The `searchparams` heuristics try various modifications of single parameters and then pairs of parameters, so variations such as moving from $p=1$ to $p=2$ or increasing iteration counts would be covered automatically. If predictions are accurate for all parameters considered by `searchparams` then `searchparams` is not being misdirected by mispredictions, although it could still be led astray because its search is only heuristic. For comparison, typical methods of automatically generating test cases ensure variations across pairs of parameters (see, e.g., [47]) and sometimes prioritize lower-cost tests (see, e.g., [38]), but aim to catch problems anywhere in the parameter space rather than more efficiently focusing on the parameters relevant to a heuristic search.

There are further risks. Even when circuits are analyzed accurately, they are not necessarily the best circuits. They could be missing known improvements; the parameter searches could have been inadequate; better circuits could be developed.

This paper's formalization already covers many ISD algorithms, but it does not claim to cover all ISD algorithms in the literature. For example, an informal analysis suggests that the low-memory algorithm in [50] is less effective than the comparably low-memory case $p'=1$ of Stern's algorithm, but this analysis has not been formalized. Further examples were mentioned in Section 4.1; see also Appendix M.

Finally, the fact that a cost metric is fully defined does not mean that it covers all costs of interest. In particular, for sorting larger and larger arrays—and for the applications of sorting inside `isd1` and `isd2` as p' and p'' increase—the cost metric in Section 2.1 is a more and more severe underestimate of real costs. See Appendix F.6.

M Other ISD cost reductions

This section presents some other circuit-cost reductions that have not been integrated into CAT.

M.1. Minor optimizations for computing $\mathcal{S}_d(A, v)$. Following the discussion in Section 5.3, in the tree for computing $\mathcal{S}_d(A, v)$, observe that there are non-root nodes $\mathcal{N}(v, A, I)$ with $d - |I| = \min(I)$ and $|I| < d$. Each descendant of such a node and itself must have at most one child, which implies that it can only lead to one leaf node

$$\mathcal{N}\Big(v, A, I \cup \big[\min(I) - 1\big]\Big) = \mathcal{N}\Big(v, A, \big(I \setminus \{\min(I)\}\big) \cup \big[\min(I)\big]\Big).$$

Therefore, we can redefine a non-root node $\mathcal{N}(s, A, I)$ with $d - |I| = \min(I)$ and $|I| < d$ as

$$\Big(v + A\big[I \setminus (\min(I)\big] + A\big[\big[\min(I)\big]\big], \big(I \setminus \{\min(\mathrm{I})\}\big) \cup \big[\min(I)\big]\Big)$$

and consider it as a leaf node. By precomputing $A\big[[i]\big]$ for $i = 0, \ldots, d-1$, each redefined node still takes only 1 vector addition to compute. As another minor optimization, the vector additions for generating the children of the root can be skipped when it is known that $v = 0$.

M.2. Computing $|I \oplus J|$ by merging sorted lists. Following the discussion in Section 5.7, to compute $|I \oplus J|$, the circuits in CAT first sort the elements in I and J together to obtain a sorted list. As I and J are represented as two sorted lists, instead of applying a sorting network on the elements, one can also use an algorithm that merges two sorted lists. One efficient option is Batcher's "odd-even merge" algorithm [70], which takes $O(d \log d)$ compare-and-exchange operations to merge two sorted lists of d elements, with a small O constant.

M.3. Finding collisions in `isd2` by merging sorted lists. As mentioned in Section 5.12, to find collisions between $S_L^{(0)}$ and $S_R^{(0)}$ (and similarly between $S_L^{(0)}$ and $\hat{S}_R^{(0)}$), the circuits in CAT sort the elements in the two sets together to form a sorted list. Instead of applying a sorting network directly, one can sort elements in the two sets separately and then merge the two sorted lists. Note that $S_L^{(0)}$ only needs to be sorted once at the beginning of each search phase. After that, for each Δ, it remains to sort $S_R^{(0)}$ and then merge the two sorted lists. This replaces D times sorting elements in $S_L^{(0)}$ and $S_R^{(0)}$ together with 1 time sorting $S_L^{(0)}$, D times sorting $S_R^{(0)}$, and D times merging $S_L^{(0)}$ with $S_R^{(0)}$.

M.4. Sorting elements in $S_R^{(0)}$ and $\hat{S}_R^{(0)}$ in `isd2`. Suppose in each search phase of `isd2`, Δ is set to (in chronological order) $\delta_0, \delta_1, \ldots, \delta_{D-1}$. Let $S_R^{(0)}(\delta_i)$ and $\hat{S}_R^{(0)}(\delta_i)$ be the sets $S_R^{(0)}$ and $\hat{S}_R^{(0)}$ for $\Delta = \delta_i$. Following the discussion in Appendix M.3, when $S_R^{(0)}(\delta_i)$ (resp. $\hat{S}_R^{(0)}(\delta_i)$) where $i > 0$ is sorted, $S_R^{(0)}(\delta_{i-1})$ (resp. $\hat{S}_R^{(0)}(\delta_{i-1})$) must have been sorted. This can be exploited to save bit operations. The following description is for $S_R^{(0)}$, but the same ideas also apply to $\hat{S}_R^{(0)}$. Given a vector $v \in \mathbb{F}_2^d$, denote by $v_{\geq i}$ and $v_{>i}$ the vectors $(v_i, \ldots, v_{d-1})$ and $(v_{i+1}, \ldots, v_{d-1})$, respectively.

This speedup is enabled by the structure of a circuit for comparing vectors. Let $v, w \in \mathbb{F}_2^d$. To compute $\mathcal{E}(v > w) = \mathcal{E}(v_{\geq 0} > w_{\geq 0})$, the circuit computes

$$\begin{aligned}&\mathcal{E}(v_{\geq d-1} > w_{\geq d-1}), \mathcal{E}(v_{\geq d-1} \neq w_{\geq d-1}),\\ &\mathcal{E}(v_{\geq d-2} > w_{\geq d-2}), \mathcal{E}(v_{\geq d-2} \neq w_{\geq d-2}), \ldots,\\ &\mathcal{E}(v_{\geq 0} > w_{\geq 0})\end{aligned}$$

sequentially. Each $\mathcal{E}(v_{\geq d-i} \neq w_{\geq d-i})$ with $i > 1$ is computed as $(v_{d-i} + w_{d-i}) \vee \mathcal{E}(v_{\geq d-i+1} \neq w_{\geq d-i+1})$, where $\vee$ indicates the OR operation. Each $\mathcal{E}(v_{\geq d-i} > w_{\geq d-i})$ with $i > 1$ is computed as

$$\Big(v_{d-i}(1 - w_{d-i})\big(1 - \mathcal{E}(v_{\geq d-i+1} \neq w_{\geq d-i+1})\big)\Big) \vee \mathcal{E}(v_{\geq d-i+1} > w_{\geq d-i+1}).$$

Apparently each of $\mathcal{E}(v_{\geq d-1} > w_{\geq d-1})$ and $\mathcal{E}(v_{\geq d-1} \neq w_{\geq d-1})$ takes only 1 bit operation to compute.

The circuits in CAT represent $S_R^{(0)}(\delta_i)$ as a variable list L_{δ_i} consisting of elements in the set. Denote by $\Pi(L_{\delta_i})$ the result of sorting L_{δ_i}. For ease of notation below, abbreviate each element (v, v', I) in L_{δ_i} as simply v. Following the discussion in Appendix M.3, each L_{δ_i} with $i > 0$ is obtained by adding u_{f_i} to each element in $\Pi(L_{\delta_{i-1}})$, where f_i is defined as the integer such that $u_{f_i} = \delta_i + \delta_{i-1}$. In this way, for $x < y$, we have

$$L_{\delta_i}[x]_{>f_i} = \Pi(L_{\delta_{i-1}})[x]_{>f_i} \leq \Pi(L_{\delta_{i-1}})[y]_{>f_i} = L_{\delta_i}[y]_{>f_i} \quad (1)$$

before and after L_{δ_i} is sorted. This implies that, for each compare-and-swap operation carried out for sorting L_{δ_i}, some bit operations can be saved by only swapping the "bottom bits", i.e., the bits of indices smaller than or equal to f_i. Also, some more bit operations can be saved whenever $\mathcal{E}(L_{\delta_i}[x] > L_{\delta_i}[y])$ is computed, as $\mathcal{E}(L_{\delta_i}[x]_{>\alpha} > L_{\delta_i}[y]_{>\alpha}) = 0$ for any $\alpha \geq f_i$.

In fact, many "$\mathcal{E}(\neq)$" values are already known and do not need to be recomputed. To see this, consider computation of $\mathcal{E}(L_{\delta_i}[x] > L_{\delta_i}[y])$ with $i \geq 1$ and $x < y$ when L_{δ_i} is sorted. To compute the value, it is necessary to obtain $\mathcal{E}(L_{\delta_i}[x]_{\geq f_i+1} \neq L_{\delta_i}[y]_{\geq f_i+1})$, which is always derived from

$$\mathcal{E}(L_{\delta_i}[x]_{\geq \ell_0-1} \neq L_{\delta_i}[y]_{\geq \ell_0-1}), \ldots, \mathcal{E}(L_{\delta_i}[x]_{\geq f_i+2} \neq L_{\delta_i}[y]_{\geq f_i+2}).$$

This means that when we compute $\mathcal{E}(L_{\delta_{i+1}}[x] > L_{\delta_{i+1}}[y])$ (when $L_{\delta_{i+1}}$ is sorted), if $f_{i+1} > f_i$, $\mathcal{E}(L_{\delta_{i+1}}[x]_{>f_{i+1}} \neq L_{\delta_{i+1}}[y]_{>f_{i+1}})$ must have been derived before. Similarly, when we compute $\mathcal{E}(L_{\delta_{i+1}}[x] > L_{\delta_{i+1}}[y])$, if $f_{i+1} < f_i$, $\mathcal{E}(L_{\delta_{i+1}}[x]_{>f_i} \neq L_{\delta_{i+1}}[y]_{>f_i})$ must have been derived before. Therefore, bit operations can be saved by reusing the "$\mathcal{E}(\neq)$" values that have been derived before. Note that this optimization enlarges the "state", i.e., the set of bits that need to be maintained simultaneously, but the size of the state is not considered in the cost metric defined in Section 2.1.

Under some conditions, comparing two vectors can be reduced to comparing the most significant bits of them. To see this, consider computation of $\mathcal{E}(L_{\delta_i}[x] > L_{\delta_i}[y])$ where $i > 0$ and $x < y$ when L_{δ_i} is sorted. Observe that under the condition that $L_{\delta_i}[x] + u_{f_i} \leq L_{\delta_i}[y] + u_{f_i}$, $L_{\delta_i}[x] > L_{\delta_i}[y]$ if and only if

- $L_{\delta_i}[x]_{>f_i} = L_{\delta_i}[y]_{>f_i}$ and
- $L_{\delta_i}[x]_{f_i} = 1, L_{\delta_i}[y]_{f_i} = 0$.

Indeed, Equation 1 shows that it is impossible to have $L_{\delta_i}[x]_{>f_i} > L_{\delta_i}[y]_{>f_i}$, and $L_{\delta_i}[x]_{\geq f_i} = L_{\delta_i}[y]_{\geq f_i}$ implies that $L_{\delta_i}[x] \leq L_{\delta_i}[y]$ under the condition $L_{\delta_i}[x] + u_{f_i} \leq L_{\delta_i}[y] + u_{f_i}$. As $L_{\delta_i}[x] =$

$\Pi(L_{\delta_{i-1}})[x]+u_{f_i}$ and $L_{\delta_i}[x] = \Pi(L_{\delta_{i-1}})[y]+u_{f_i}$ before any compare-and-swap operation is carried out, we must have $L_{\delta_i}[x] + u_{f_i} \leq L_{\delta_i}[y] + u_{f_i}$ if the pair of entries $(L_{\delta_i}[x], L_{\delta_i}[y])$ has not been used in any compare-and-swap operation. To make use of this, maintain a vector $v \in \mathbb{F}_2^{|L_{\delta_i}|}$ when L_{δ_i} is sorted, such that $v_x = 1$ if and only if $L_{\delta_i}[x]$ has not been used in any compare-and-swap operation. To figure out whether $L_{\delta_i}[x] > L_{\delta_i}[y]$, if $(v_x, v_y) = (1, 1)$, simply figure out whether $L_{\delta_i}[x]_{\geq f_i} > L_{\delta_i}[y]_{\geq f_i}$. Note that maintaining v is free in this cost metric, as whether an entry is used is independent of the data being sorted.

N Previous analyses of ISD effectiveness

It is easy to find clear high-level statements of ISD algorithms in the ISD literature, and clear explanations of the most important bottlenecks in the algorithms. However, cryptography uses much more precise evaluations of attack costs; see Appendix A.

This appendix reviews various quantitative statements in the previous literature regarding the ISD cost/probability ratio. All of the statements sound reasonably precise, but a closer look shows that none of the statements have clearly defined semantics. Models of computation, cost metrics, and choices of subroutines are not pinpointed; they are only loosely constrained.

Often there are variations in security levels reported for the same (n, k, t), as the examples below illustrate. Certainly there are cases where algorithm B in paper Y is better than algorithm A in paper X, such as Leon's algorithm outperforming Prange's original algorithm. There are also cases where Y is assigning lower costs than X to the same operations. There are also known cases of errors one way or the other, such as X overestimating the cost of A, or Y underestimating the cost of B, or X underestimating the cost of A but Y more severely underestimating the cost of B. It is labor-intensive to disentangle these effects: readers have to manually check each step of each algorithm analysis. ISD software is sometimes provided, but uses different cost metrics from the algorithm analyses and, as in Appendix C.1, has limited value in helping readers catch errors in analyses.

These difficulties are not specific to ISD. This paper uses ISD as a case study, but the core problems are much broader, as is the approach that this paper takes to address these problems.

N.1. 1978 McEliece and 1987 Adams–Meijer. [84] says that "one expects a work factor of $k^3 \cdot (1 - \frac{t}{n})^{-k}$", and plugs in $(n, k, t) = (1024, 524, 50)$ as an example, obtaining "about $10^{19} \approx 2^{65}$". The "work factor" cost metric is undefined.

This is preceded by a statement that "the amount of work involved in solving the k simultaneous equations in k unknown is about k^3". The choice of linear-algebra subroutine here is undefined. The simplest linear-algebra subroutines use $\Theta(k^3)$ field operations, but operations for small fields can easily be batched if the model of computation allows word operations, and a logarithmic factor can be saved if the model of computation allows random access. Also, as mentioned in [4], smaller asymptotic exponents than 3 were already known (from [39], which eliminated the failure cases in [115]), although this is not necessarily important for the sizes of k used in cryptography.

At a higher level, the algorithm description says "select k of the n coordinates randomly". Presumably this means uniformly at random, but then the corresponding $k \times k$ matrix is usually not invertible (i.e., the coordinates are usually not an information set), so the linear-algebra problem is actually to enumerate a variable-dimension solution space, not just to find one solution. One cannot tell, from the level of description in [84], whether the costs of enumerating and checking solutions were evaluated as fitting within k^3 *on average* (certainly they do not in the worst case), or were simply neglected.

Similarly, [4] says "The work factor for this attack can be calculated as follows", without defining the "work factor" cost metric. The algorithm statement in [4] is not exactly the same as in [84]: it explicitly hypothesizes that the relevant $k \times k$ matrix is invertible, and inverts the matrix, presumably skipping the iteration if the matrix turns out not to be invertible. The probability of invertibility, approximately 30% for uniform random matrices, is ignored in the analysis in [4].

Regarding the number of iterations, $(1 - \frac{t}{n})^{-k}$ in [84] is a much worse approximation than $\binom{n}{t}/\binom{n-k}{t}$ (used in, e.g., [4]). These quantities are approximately $2^{37.84}$ and $2^{53.61}$ respectively for $(n, k, t) = (1024, 524, 50)$.

For comparison, the version of Prange's ISD algorithm in CAT is fully defined all the way down through the model of computation and cost metric, so in particular it resolves the ambiguities regarding the cost of linear algebra; CAT's predictors of cost and success probability are also fully defined, and compared directly to simulations (see, e.g., Figure 4). As one would expect from the details of the algorithm, the simplified formula $k^3\binom{n}{t}/\binom{n-k}{t}$ is a reasonable, although not perfect, prediction of the cost/probability ratio. For example, this quantity is approximately $2^{85.85}$ for $(n, k, t) = (1284, 1020 - 1, 24)$ and $2^{338.04}$ for $(n, k, t) = (8192, 6528 - 1, 128)$, close to the $2^{85.99}$ and $2^{338.10}$ in Table 1; the subtraction of 1 from k accounts for $\mathtt{FW} = 1$.

N.2. 1988 Lee–Brickell. [75] says that its new algorithm "reduces the work factor significantly (factor of 2^{11} for the commonly used example of $n = 1024$ Goppa code case)". The "work factor" cost metric is undefined.

The details of the analysis in [75] count two main algorithm steps: linear algebra, where the "work factor" is claimed to be "approximately αk^3 with small α", and a combinatorial search, where the "work factor" for each testing step is claimed to be "approximately βk with small β".

The high-level point is that one can amortize the cost of linear algebra across many testing steps "for any reasonable value of α and β". This statement is reasonably robust against variations in the choice of cost metrics: seeing that the Lee–Brickell algorithm outperforms Prange's algorithm does not require a very detailed algorithm analysis.

N.3. 1998 Canteaut–Chabaud. When algorithm performance is converging, recognizing speedups inherently requires more and more precision. [41] gives an algorithm with several parameters, combining the approaches of Prange, Omura, Lee–Brickell, Leon, and Stern; and says that it gives "a very precise analysis of the complexity of this algorithm which enables us to optimize the parameters it depends upon".

[41] then gives an exact formula for "the average number of elementary operations performed at each iteration", and [41, "Proposition 7"] claims an exact formula for the "overall work factor". However, there is no definition of "work factor", or of which operations are allowed as "elementary operations".

The reader can try to deduce constraints on the concept of "elementary operations" by inspecting details of the algorithm analysis. For example, the statement "We need $K(p\binom{k/2}{p} + 2^\sigma)$ more operations to performthe [*sic*] dynamic memory allocation where K is the size of a computer word ($K = 32$ or 64)" is in the context of a "hash table with 2^σ entries" storing $p\binom{k/2}{p}$ values, and appears to be making an ad-hoc assumption that the number of "operations" to manage the hash table is the "size of a computer word" times the number of entries plus values in the table.

Note that, structurally, allowing ad-hoc assumptions in algorithm analyses means that algorithm B is free to make ad-hoc assumptions producing smaller "operation" counts than algorithm A, even when a comparison in a well-defined cost metric would show that B is slower than A. It is qualitatively clear that [41] outperforms previous algorithms, but quantifying this requires clarity regarding the cost metric.

N.4. 2008 Bernstein–Lange–Peters. [29] is structured to support direct comparisons: it quotes previous operation counts (e.g., "Stern says that reduction involves about $(1/2)(n-k)^3 + k(n-k)^2$ bit operations"), explains how those counts appear to have been calculated from algorithm steps ("To understand this formula, observe that the first column requires $\leq n-k$ reductions" etc.); and then explains how the calculation changes when some steps are eliminated (e.g., "the number of reductions in a typical column is only about $(n-k-1)/2$").

This micro-comparison approach again makes qualitatively clear that there are speedups. However, it does not quantify the overall speedups in any particular cost metric: in particular, it still does not define "bit operations".

[29] also reports software performance, but notes that "optimizing CPU cycles is different from, and more difficult than, optimizing the simplified notion of 'bit operations' " used in the predictions, as mentioned in Section 3. Software is easy to measure, but these measurements are not directly comparable to the bit-operation predictions.

N.5. Interlude: Challenges. The software described in [29] was used in 2008 to break a challenge for McEliece's original parameters $(1024, 524, 50)$, and was used in 2023 to set a new record in the series of challenges from [73], breaking a challenge for the size $(1347, 1047, 25)$ mentioned in Section 6; see [23].

This suggests that ISD algorithms have not improved much since 2008. However, there are several reasons for caution regarding the general idea of using challenges to measure improvements in ISD algorithms.

When a new record is set in a challenge, the record might come from better algorithms, or from continued improvements in chip technology, or from more money being spent on chips, or—for high-variance computations, such as AES key search or ISD—simply being lucky. A sufficiently large change in algorithm cost (such as [43] breaking SIKE) will be easily visible as a sudden jump in records, but obviously the improvements in ISD have been much smaller than this.

The improvements in ISD algorithms *after* the introduction of `isd1` in the 1980s are almost invisible at the sizes of recently broken challenges. For example, the `isd1` row in Table 1 with $\mathsf{RE} = 1$ and $p' = 2$ says 71.66, while the smallest number in the column is 70.90. This difference is so small that trying to detect it from a single challenge run is statistically invalid. Meanwhile the computer power available to public researchers has increased by a much larger factor over the same period. One expects larger and larger challenges to be broken whether or not there are any algorithmic improvements.

Challenges also have worrisome second-order effects. It is important to recognize algorithm speedups even when the speedups are small (see Appendix A.5), and running enough trials can reliably detect small differences, but challenges instead encourage computer power to be spent on a single trial at the largest affordable size. Furthermore, what is broken in a challenge is at most what can be broken by public researchers today, whereas large-scale attackers have much more computer power today and will have even more computer power in the future.

N.6. 2017 Classic McEliece. The Classic McEliece submission in 2017 [25] to the NIST Post-Quantum Cryptography Standardization Project reviews the proposal of $n = 6960$ from [29] (which says that this was designed to maximize security for keys limited to 2^{20} bytes) and then says that "subsequent ISD variants have reduced the number of bit operations considerably below 2^{256}". The concept of "bit operations" used here is not defined.

Subsequent versions of the Classic McEliece submission make the same "considerably below 2^{256}" comment, again without defining "bit operations". The latest version [8] also says this is consistent with a $2^{246.6}$ number produced by the estimator in [55]; see below.

The submission also says "We expect that switching from a bit-operation analysis to a cost analysis will show that this parameter set is more expensive to break than AES-256 pre-quantum and much more expensive to break than AES-256 post-quantum". The preceding "cost" comments refer to "hardware", indicating that this is a statement about real costs; certainly "cost" is not given a mathematical definition.

N.7. 2019 Baldi–Barenghi–Chiaraluce–Pelosi–Santini. [13] says that it provides "exact formulas for the time complexity" for a variety of ISD algorithms. For example, [13, Proposition 6] states the "time complexity of May–Meurer–Thomae". However, the "time" cost metric is undefined.

[13, Table 4] summarizes concrete costs computed from the same formulas. In the table, "MMT" is an evident outlier, having much lower cost than the other ISD algorithms. For example, for $(3488, 2720, 64)$, the 8 non-quantum numbers listed in the table include $2^{152.51}$ for "St", $2^{149.91}$ for "BJMM", and just $2^{118.61}$ for "MMT". (For comparison, Table 1 lists 156.96 for `isd1`, and lists 155.38 for `isd2` with $p' = 2p''$ and $C = 0$, which is essentially the 2011 MMT algorithm.)

In 2021, NIST [95] wrote "If the analysis is correct, it seems like this could threaten not just some of the McEliece parameters, but also some of the parameters of the other code-based schemes."

In reply, Kirshanova [69] wrote that "The conclusion that BJMM algorithm is worse than MMT is incorrect because MMT is a special case of BJMM". However, as [95] illustrates, non-experts did not find this obvious from the previous literature.

[69] also reported an inability to "reproduce the exact bit complexities" from [13]; and then [55, Appendix A] pointed out a specific factor missing inside the MMT formulas in [13].

[69] includes further estimates, namely $2^{133.61}$ for MMT and $2^{127.12}$ for BJMM, with the explicit caveat that these numbers are "likely to be underestimates, as poly(n) factors are ignored"—i.e., the numbers were counting only *some* of the operations inside attacks. For comparison, the smallest number in this column of Table 1 is 150.59, and the smallest number for $p' = 2p''$ and $C = 0$ is 155.38. The gap between 155.38 and 150.59 is similar to the gap between 133.61 and 127.12, but the gap between 127.12 and 150.59 makes an obvious difference for, e.g., NIST comparing to the cost of breaking AES-128.

N.8. 2022 Esser–Bellini. [55] claimed to "analyze the complexity of all algorithms in a unified and practical model giving a fair comparison and concrete hardness estimations"; claimed that the analysis produced "formulas for the concrete complexity to solve the syndrome decoding problem"; and claimed that the software from [55] allowed "for an effortless recomputation of our results".

The estimator from [55] consists of cost-prediction formulas. Since the estimator is open-source, it is easy to ask a computer to convert these formulas into concrete cost predictions for specific problem sizes. However, this provides no assurance that the formulas correctly compute "the concrete complexity" of any particular attacks.

Furthermore, evaluating whether formulas correctly compute costs requires a definition of the cost metric. A reader who searches for the definition of the "unified and practical model" in [55] will not find a definition. There are merely assertions regarding the costs of various algorithm steps in an unspecified model of computation, as in earlier papers. See, e.g., the claim from [55] reviewed below regarding the "cost" of finding collisions.

For $(3488, 2720, 64)$, [55, Table 2] reports "bit security estimates" of 2^{151} for "Stern" and 2^{142} for "BJMM". For comparison, recall that [13] said $2^{152.51}$ for "St" and $2^{149.91}$ for "BJMM". Table 1 says 156.96 for `isd1` and 150.59 for `isd2`.

These numbers are not very far apart, and all of them might seem comfortably beyond the 2^{111} bit operations carried out worldwide by Bitcoin in 2022. However, the numbers matter for cryptosystems designed for long-term security. In the NIST Post-Quantum Cryptography Standardization Project, NIST required cryptosystems to be *at least* as secure as AES-128, estimated AES-128 as requiring 2^{143} bit operations to break (see Appendix D), and recently made standardization decisions based on very close comparisons to AES-128 (see Appendix A.3). In this context, one cannot ignore the difference between 142 and 150.

It is natural to ask whether the lower numbers in [55], compared to Table 1, come from [55] using different, better optimized, "Stern" and "BJMM" algorithms, or instead from [55] missing important components of attack costs. As an apparently critical example of the latter, [55, Formula (1)] uses $2L + L^2/2^\ell$ as the "cost" of finding all (u, v, u', v') with

- (u, v) from a given length-L list of pairs,
- (u', v') from another given length-L list of pairs, and
- $v = v'$, where v and v' have ℓ bits.

This "cost" is far below the cost of any known circuit for the same collision-finding problem. One would have to multiply by the number of bits in each vector simply to reach the number of bits of input and output; more importantly, there are many intermediate bit operations, as illustrated by the sorting circuits in Section 5.

If there were a clear definition of the "unified and practical model" in [55] then one could determine how much of this underestimate comes from inaccuracies built into the model and how much comes from inaccuracies in analyzing costs within that model. There are some comments in [55] regarding cost metrics, such as "we measure all running times in vector operations in $\mathbb{F}_2^m$", but this does not make clear which "vector operations" are counted, and how these "operations" allow collision-finding at "cost" just 1 per input and 1 per output. Presumably such powerful "operations" can also be exploited to reduce the "cost" of other algorithms, as in Appendix D.

The statement "we measure all running times in vector operations in $\mathbb{F}_2^m$" in [55] was reported in [8] as a reason that the numbers in [55] were underestimates ("the underlying estimator from [33] counts each vector operation as just 1 operation" so "should be expected to be superseded by larger numbers from future estimators that count bit operations"). The situation is, however, more complicated than this: the software from [55] multiplied the "cost"/"running time" formulas from [55] by n to obtain the "bit security" numbers in [55]. This fudge factor overstates most of the vector lengths used in the algorithm, while it still does not account for the intermediate bit operations needed for collision-finding. The numbers in [55] generally end up several bits below the numbers in Table 1, although the gap is narrower for small p, p', p'' for reasons pointed out below.

N.9. 2022 Esser–Bellini, continued. There are also alternative numbers in [55], for example indicating that 2^{142} jumps to 2^{156} if one switches to a "cube-root model". This "model" multiplies the previous "cost" by "$\sqrt[3]{M}$", so it inherits all of the definitional problems surrounding the concept of "cost" in [55].

A closer look at the underlying estimator output also shows the estimator saying that this "model" compresses differences between algorithms, for example compressing the Stern-vs.-BJMM gap for $n = 3488$ from 9 bits to 0.2 bits. See [8, "Guide for security reviewers", Table 1].

Qualitatively, these effects are not surprising. It has been well known for many years that accounting for long-distance communication costs changes the exponent of sorting and many other large-memory algorithms; see Appendix F.6 for references. Simply multiplying by "$\sqrt[3]{M}$" has a similar effect. Also, `isd2` relies much more than `isd1` does on using large amounts of memory, as illustrated by the p'' choices for `isd2` in Table 1; assigning higher costs to memory encourages smaller values of p' and p'', reducing the `isd2` benefit.

Quantitatively, a problem with the numbers in [55] for small p, p', p'' is that the algorithms in [55] make no use of random walks: each algorithm iteration uses a full echelon-form computation. This explains why the estimator from [55] recommends, e.g., $p' = 2$ for Stern and $p'' = 3$ for BJMM in the case $(n, k, t) = (1284, 1020, 24)$,

whereas the best parameters in Table 1 have $p' = 1$ for `isd1` and $p'' = 1$ for `isd2`. The estimator from [55] is assigning very low costs to memory-intensive collision-finding, while choosing algorithms that make linear algebra unnecessarily expensive.

No matter how small p, p', p'' are, an ISD iteration carries out matrix operations, so it requires communication across a considerably larger circuit than, e.g., an AES key-search iteration. A realistic evaluation of the costs of ISD requires realistically modeling these communication costs *and* properly optimizing low-memory algorithms within this model. Obviously "$\sqrt[3]{M}$" was designed for simplicity rather than for accuracy, and the necessary low-memory optimizations are missing from [55].

N.10. How well do CPU timings predict large-scale attack costs? A structurally different approach to predicting ISD costs is taken in another paper, 2021 Esser–May–Zweydinger [57], which claims to provide "precise bit-security estimates for code-based cryptography such as McEliece". In particular, the paper claims that, with "the MMT/BJMM algorithm", McEliece with parameters $(3488, 2720, 64)$ is 1.17 bits harder to break than AES-128. Given NIST's estimate of AES-128 as costing 2^{143} bit operations to break (see Appendix D), [57] would appear to be claiming 2^{144} bit operations.

However, no definition of "bit security" is specified in [57]. The computation of 1.17 instead comes from the following chain of calculations:

- [57] reports that its $(1284, 1020, 24)$ attack software takes on average "37.47" days on a cluster "consisting of two nodes, each one equipped with 2 AMD EPYC 7742 processors and 2 TB of RAM", in total 256 cores.
- [57] reports $2.16 \cdot 10^9$ AES-128 encryptions/second on the same cluster. In other words, the $n = 1284$ attack software took the same time on the cluster as $2^{52.63}$ AES-128 encryptions.
- Regarding memory-access costs, [57] uses a curve-fitting technique to conclude that "a logarithmic access cost most accurately models our experimental data".
- [57] uses an ISD estimator with logarithmic access cost to conclude that $(3488, 2720, 64)$ is $2^{76.54}$ times more difficult to break than $(1284, 1020, 24)$, and thus would take the same time "on our hardware" as $2^{129.17}$ AES-128 encryptions, which is then compared to 2^{128} AES-128 encryptions.

The attacks against cryptographic parameters considered in [57] require roughly 2^{100} bits of storage, so it is not correct that they could run on the same cluster (even if the cluster could last this long). Presumably the intention was instead to extrapolate to the capabilities of an attacker building hardware at a much larger scale.

The second step in the above chain of calculations overestimates the real-world price-performance ratio of breaking AES-128 by five orders of magnitude. Quantitatively, each 64-core EPYC 7742 CPU has $32 \cdot 10^9$ transistors according to [108], so each CPU core has $0.5 \cdot 10^9$ transistors. According to [58, Zen2 tables, "AESENC"], each of these CPU cores carries out at best two parallel AES rounds per cycle. Each round uses a few thousand bit operations (see Appendix D.4), accounting for only a tiny fraction of the transistors in the CPU. The attacker will obtain a much better price-performance ratio using dedicated key-search circuits, with most transistors performing cipher operations at each moment, with only minor overheads for key selection and comparison.

One might think that a special-purpose circuit dedicated to parallel bit operations would catch fire. To see that this is incorrect, consider Bitcoin-mining ASICs (also mentioned in Appendix F):

- According to [10], the Antminer S17—which uses the same 7nm technology as the EPYC 7742 CPU—carries out 56 terahashes/second at 2520 watts, i.e., $45 \cdot 10^{-12}$ joules per hash. If these hashes are full Bitcoin hashes, double SHA-256, then each hash is roughly 2^4 times as expensive as AES-128 encryption, so a similar AES-128 attack machine would use roughly $3 \cdot 10^{-12}$ joules per AES-128 encryption.
- For comparison, the power consumption of the cluster in [57] is not reported but presumably is roughly 1000 watts, so the reported $2.16 \cdot 10^9$ AES-128 encryptions per second correspond to roughly $500000 \cdot 10^{-12}$ joules per AES-128 encryption.

This does not mean that special-purpose hardware is five orders of magnitude more efficient than mass-market computers for *all* computations. In particular, mass-market computers spend much more hardware (and energy) on RAM.

A large-scale attacker targeting $(3488, 2720, 64)$ with ISD would also try to build special-purpose hardware for that, but cannot hope to do better than indicated by the number of bit operations in a Boolean-circuit model; see Appendix F.5. Furthermore, as the parameters p, p', p'' grow, the attacker would be faced with increasingly severe communication costs. The curve-fitting procedure in [57] appears to have considered only experiments within a single level of the CPU's memory hierarchy, missing the larger changes in memory-access costs on the same CPU between L1 cache, L2 cache, L3 cache, and DRAM. See [2].

If [57] had selected Bitcoin-mining ASICs as its baseline then its final 2^{144} would have jumped to about 2^{160}. If it had chosen different boundaries in the memory hierarchy then its curve-fitting procedure could have produced an even larger jump, depending on tiny measurement details and on arbitrary choices of scaling functions. Given the lack of a definition of "bit security", the "bit-security estimates" obtained by any of these procedures would not meet the requirement of falsifiability; the same comment applies to the estimates in [57].

N.11. The future. This paper provides a framework that enforces links between a clearly defined general-purpose model of computation, a clearly defined general-purpose cost metric, clearly defined attack algorithms, and clearly defined predictions of attack effectiveness. Prediction errors might still occur (see Appendix L), but they cannot hide behind ambiguities in the meaning of what is being predicted.

It would be interesting, although challenging, to add a more realistic circuit-layout model. As noted in Appendix F.6, this would make more difference for lattice attacks than for ISD. It is in any case clear that models should not be created ad-hoc for each attack: these are the foundations of algorithm analysis, and comparability requires these foundations to be shared.