Non-uniform cracks in the concrete: the power of free precomputation

Daniel J. Bernstein[1,2] and Tanja Lange[2]

[1] Department of Computer Science
University of Illinois at Chicago, Chicago, IL 60607–7053, USA
djb@cr.yp.to
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands
tanja@hyperelliptic.org

Abstract. AES-128, the NIST P-256 elliptic curve, DSA-3072, RSA-3072, and various higher-level protocols are frequently conjectured to provide a security level of 2^{128}. Extensive cryptanalysis of these primitives appears to have stabilized sufficiently to support such conjectures.

In the literature on provable concrete security it is standard to define 2^b security as the nonexistence of high-probability attack algorithms taking time $\le 2^b$. However, this paper provides overwhelming evidence for the existence of high-probability attack algorithms against AES-128, NIST P-256, DSA-3072, and RSA-3072 taking time considerably below 2^{128}, contradicting the standard security conjectures.

These attack algorithms are not realistic; do not indicate any actual security problem; do not indicate any risk to cryptographic users; and do not indicate any failure in previous cryptanalysis. Any actual use of these attack algorithms would be much more expensive than the conventional 2^{128} attack algorithms. However, this expense is not visible to the standard definitions of security. Consequently the standard definitions of security fail to accurately model actual security.

The underlying problem is that the standard set of algorithms, namely the set of algorithms taking time $\le 2^b$, fails to accurately model the set of algorithms that an attacker can carry out. This paper analyzes this failure in detail, and analyzes several ideas for fixing the security definitions.

Keywords: provable security, concrete security, algorithm cost metrics, non-uniform algorithms, non-constructive algorithms

Full version. See http://cr.yp.to/nonuniform.html for the full version of this paper. Appendices appear only in the full version.

This work was supported by the National Science Foundation under grant 1018836, by the Netherlands Organisation for Scientific Research (NWO) under grant 639.073.005, and by the European Commission under Contract ICT-2007-216676 ECRYPT II. Permanent ID of this document: 7e044f2408c599254414615c72b3adbf. Date: 2013.09.10.

1 Introduction

The Basic Principles of Modern Cryptography ...

Principle 1—Formulation of Exact Definitions

One of the key intellectual contributions of modern cryptography has been the realization that formal definitions of security are essential *prerequisites for the design, usage, or study of any cryptographic primitive or protocol.* —Katz and Lindell [53]

In this paper we will show that CBC MAC construction is secure if the underlying block cipher is secure. To make this statement meaningful we need first to discuss what we mean by security in each case.
—Bellare, Kilian, and Rogaway [12, Section 1.2]

Why do we believe that AES-CBC-MAC is secure? More precisely: Why do we believe that an attacker limited to 2^{100} bit operations, and 2^{50} message blocks, cannot break AES-CBC-MAC with probability more than 2^{-20}?

The standard answer to this question has three parts. The first part is a concrete definition of what it means for a cipher or a MAC to be secure. We quote from the classic paper [12, Section 1.3] by Bellare, Kilian, and Rogaway: the PRP-“insecurity” of a cipher such as AES (denoted “$\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(q',t')$”) is defined as the “maximum, over all adversaries restricted to q' input-output examples and execution time t', of the ‘advantage’ that the adversary has in the game of distinguishing [the cipher for a secret key] from a random permutation.” The PRF-insecurity of m-block AES-CBC-MAC (denoted “$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CBC}^m\text{-}\mathrm{AES}}(q,t)$”) is defined similarly, using a uniform random function rather than a uniform random permutation.

The second part of the answer is a concrete security theorem bounding the insecurity of AES-CBC-MAC in terms of the insecurity of AES, or more generally the insecurity of F-CBC-MAC in terms of the insecurity of F for any ℓ-bit block cipher F. Specifically, here is the main theorem of [12]: “for any integers $q, t, m \geq 1$,

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CBC}^m\text{-}F}(q,t) \leq \mathbf{Adv}^{\mathrm{prp}}_{F}(q',t') + \frac{q^2m^2}{2^{l-1}}$$

where $q' = mq$ and $t' = t + O(mql)$.” One can object that the O constant is unspecified, making this theorem meaningless as stated for any specific q, t, m values; but it is easy to imagine a truly concrete theorem replacing $O(mql)$ with the time for mql specified operations.

The third part of the answer is a concrete conjecture regarding the security of AES. NIST's call for AES submissions [66, Section 4] identified “the extent to which the algorithm output is indistinguishable from [the output of] a [uniform] random permutation” as one of the “most important” factors in evaluating candidates; cryptanalysts have extensively studied AES without finding any worrisome PRP-attacks; it seems reasonable to conjecture that no dramatically better attacks exist. Of course, this part of the story depends on the details of

AES; analogous conjectures regarding, e.g., DES would have to be much weaker. For example, Bellare and Rogaway in [16, Section 3.6] wrote the following:

"For example we might conjecture something like:

$$\mathbf{Adv}^{\text{prp-cpa}}_{\text{DES}}(A_{t,q}) \le c_1 \cdot \frac{t/T_{\text{DES}}}{2^{55}} + c_2 \cdot \frac{q}{2^{40}}$$

... In other words, we are conjecturing that the best attacks are either exhaustive key search or linear cryptanalysis. We might be bolder with regard to AES and conjecture something like

$$\mathbf{Adv}^{\text{prp-cpa}}_{\text{AES}}(B_{t,q}) \le c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + c_2 \cdot \frac{q}{2^{128}}.\text{"}$$

One can again object that the c_1 and c_2 are unspecified here, making these conjectures non-concrete and unfalsifiable as stated. A proper concrete conjecture would specify, e.g., $c_1 = c_2 = 3$. One can also quibble that the T_{DES} and T_{AES} factors do not properly account for inner-loop speedups in exhaustive key search (see, e.g., [27]), that $q/2^{40}$ is a rather crude model of the success probability of linear cryptanalysis, etc., but aside from such minor algorithm-analysis details the conjectures seem quite reasonable.

This AES security conjecture (with small specified c_1 and c_2) says, in particular, that the attacker cannot PRP-break AES with probability more than 2^{-21} after 2^{50} cipher outputs and 2^{100} bit operations. The CBC-MAC security theorem (with small specified O) then says that the same attacker cannot PRF-break AES-CBC-MAC with probability more than 2^{-20}.

Of course, this answer does not *prove* that AES-CBC-MAC is secure; it relies on a conjecture regarding AES security. Why not simply conjecture that AES-CBC-MAC is secure? The answer is scalability. It is reasonable to ask cryptanalysts to intensively study AES, eventually providing confidence in the security of AES, while it is much less reasonable to ask cryptanalysts to intensively study AES-CBC-MAC, AES-OMAC, AES-CCM, AES-GCM, AES-OCB, and hundreds of other AES-based protocols. Partitioning the AES-CBC-MAC security conjecture into an AES security conjecture and a CBC-MAC security proof drastically simplifies the cryptanalyst's job.

The same three-part pattern has, as illustrated by Appendix L (in the full version), become completely standard throughout the literature on concrete "provable security". First part: The insecurity of X — where X is a primitive such as AES or RSA, or a higher-level protocol such as AES-CBC-MAC or RSA-PSS — is defined as the maximum, over all algorithms A ("attacks") that cost at most C, of the probability (or advantage in probability) that A succeeds in breaking X. This insecurity is explicitly a function of the cost limit C; typically C is separated into (1) a time limit t and (2) a limit q on the number of oracle queries. Note that this function depends implicitly on how the "cost" of an algorithm is defined.

Often "the (q, t)-insecurity of X is at most ϵ" is abbreviated "X is (q, t, ϵ)-secure". Many papers prefer the more concise notation and do not even mention the insecurity function. We emphasize, however, that this is merely a superficial change in notation, and that both of the quotes in this paragraph refer to exactly the same situation: namely, the nonexistence of algorithms that cost at most (q, t) and that break X with probability more than ϵ.

Second part: Concrete "provable security" theorems state that the insecurity (or security) of a complicated object is bounded in terms of the insecurity (or security) of a simpler object. Often these theorems require restrictions on the types of attacks allowed against the complicated object: for example, Bellare and Rogaway in [14] showed that RSA-OAEP has similar security to RSA against generic-hash attacks (attacks in the "random-oracle model").

Third part: The insecurity of a well-studied primitive such as AES or RSA-1024 is conjectured to match the success probability of the best attack known. For example, Bellare and Rogaway, evaluating the concrete security of RSA-FDH and RSA-PSS, hypothesized that "it takes time $Ce^{1.923(\log N)^{1/3}(\log\log N)^{2/3}}$ to invert RSA"; Bellare, evaluating the concrete security of NMAC-h and HMAC-h, hypothesized that "the best attack against h as a PRF is exhaustive key search". See [15, Section 1.4] and [7, Section 3.2]. These conjectures seem to precisely capture the idea that cryptanalysts will not make significant further progress in attacking these primitives.

1.1. Primary contribution of this paper. Our primary goal in this paper is to convincingly undermine all of the standard security conjectures reviewed above. Specifically, Sections 2, 3, 4, and 5 show — assuming standard, amply tested heuristics — that there *exist* high-probability attacks against AES, the NIST P-256 elliptic curve, DSA-3072, and RSA-3072 taking considerably less than 2^{128} time. In other words, the insecurity of AES, NIST P-256, DSA-3072, and RSA-3072, according to the standard concrete-security definitions, reaches essentially 100% for a time bound considerably below 2^{128}. The conjectures by Bellare and Rogaway in [15, Section 1.4], [16, Section 3.6], [7, Section 3.2], etc. are false for every reasonable assignment of the unspecified constants.

The same ideas show that there *exist* high-probability attacks against AES-CBC-MAC, RSA-3072-PSS, RSA-3072-OAEP, and thousands of other "provably secure" protocols, in each case taking considerably less than 2^{128} time. It is not clear that similar attacks exist against *every* such protocol in the literature, since in some cases the security reductions are unidirectional, but undermining these conjectures also means undermining all of the security arguments that have those conjectures as hypotheses.

We do not claim that this reflects any actual security problem with AES, NIST P-256, DSA-3072, and RSA-3072, or with higher-level protocols built from these primitives. On the contrary! Our constructions of these attacks are very slow; we conjecture that any *fast* construction of these attacks has negligible probability of success. Users have nothing to worry about.

However, the standard metrics count only the cost of running the attack, not the cost of finding the attack in the first place. This means that there is a very large gap between the actual insecurity of these primitives and their insecurity according to the standard metrics.

This gap is not consistent across primitives. We identify different gaps for different primitives (for example, the asymptotic exponents for high-probability attacks drop by a factor of 1.5 for ECC and a factor of only 1.16 for RSA), and we expect that analyzing more primitives and protocols in the same way will show even more diversity. In principle a single attack is enough to illustrate that the standard definitions of security do not accurately model actual security, but the quantitative variations from one attack to another are helpful in analyzing the merits of ideas for fixing the definitions. It is of course also possible that the gaps for the primitives we discuss will have to be reevaluated in light of even better attacks.

1.2. Secondary contribution of this paper (in the full version). Our secondary goal in this paper is to propose a rescue strategy: a new way to define security — a definition that restores, to the maximum extent possible, the attractive three-part security arguments described above.

All of the gaps considered in this paper come from errors in quantifying feasibility. Each of the high-probability attacks presented in this paper (1) has a cost t according to the standard definitions, but (2) is obviously infeasible, even for an attacker able to carry out a "reasonable" algorithm that costs t according to the same definitions. The formalization challenge is to say exactly what "reasonable" means. Our core objective here is to give a new definition that accurately captures what is actually feasible for attackers.

This accuracy has two sides. First, the formally defined set of algorithms must be large enough. Security according to the definition does not imply actual security if the definition ignores algorithms that are actually feasible. Second, the formally defined set of algorithms must be small enough. One cannot conjecture security on the basis of cryptanalysis if infeasible attacks ignored by cryptanalysts are misdeclared to be feasible by the security definition.

We actually analyze four different ideas for modifying the notion of feasibility inside existing definitions:

– Appendix B.2: switching the definitions from the RAM metric used in [12] to the NAND metric, an "alternative" mentioned in [12];
– Appendix B.3: switching instead to the AT metric, a standard hardware-design metric formally defined by Brent and Kung in [29] in 1981;
– Appendix B.4: adding constructivity to the definitions, by a simple trick that we have not seen before (with a surprising spinoff, namely progress towards formalizing collision resistance); and
– Appendix B.5: adding uniformity (families) to the definitions.

Readers unfamiliar with the RAM, NAND, and AT metrics should see Appendix A (in the full version) for a summary and pointers to the literature.

The general idea of modifying security definitions, to improve the accuracy with which those definitions model actual security, is not new. A notable example is the change from the algorithm cost metric used in [11], the original Crypto '94 version of [12], to a more complicated algorithm cost metric used in subsequent definitions of security; readers unfamiliar with the details should see Appendix A for a review. The attacks in this paper show that this modification was not enough, so we push the same general idea further, analyzing the merits of the four modifications listed above. It is conceivable that this general idea is not the best approach, so we also analyze the merits of two incompatible approaches: (Appendix B.1) preserving the existing definitions of security; (Appendix B.7) trying to build an alternate form of "provable security" *without* definitions of security.

Ultimately we recommend the second and third modifications (AT and constructivity) as producing much more accurate models of actual feasibility. We also recommend refactoring theorems (see Appendix B.6) to simplify further changes, whether those changes are for even better accuracy or for other reasons. We recommend against the first and fourth modifications (NAND and uniformity). Full details of our analysis appear in Appendix B; the NAND and AT analyses for individual algorithms appear in Sections 2, 3, 4, and 5. Appendix Q (in the full version) is a frequently-asked-questions list, serving a role for this paper comparable to the role that a traditional index serves for a book.

Our recommended modifications have several positive consequences. Incorrect conjectures in the literature regarding the concrete security of primitives such as AES can be replaced by quite plausible conjectures using the new definitions. Our impression is that *most* of the proof ideas in the literature are compatible with the new definitions, modulo quantitative changes, so *most* concrete-security theorems in the literature can be replaced by meaningful concrete-security theorems using the new definitions. The conjectures and theorems together will then produce reasonable conclusions regarding the concrete security of protocols such as AES-CBC-MAC.

We do not claim that *all* proofs can be rescued, and it is even possible that some theorems will have to be abandoned entirely. Some troublesome examples have been pointed out by Koblitz and Menezes in [**55**] and [**56**]. Our experience indicates, however, that such examples are unusual. For example, there is nothing troublesome about the CBC-MAC proof or the FDH proof; these proofs simply need to be placed in a proper framework of meaningful definitions, conjectures, and theorem statements.

1.3. Priority dates; credits; new analyses. On 20 March 2012 we publicly announced the trouble with the standard AES conjectures; on 17 April 2012 we publicly announced the trouble with the standard NIST P-256, DSA-3072, and RSA-3072 conjectures. The low-probability case of the AES trouble was observed independently by Koblitz and Menezes and announced earlier in March 2012; further credits to Koblitz and Menezes appear below. We are not aware of previous publications disputing the standard concrete-security conjectures.

Our attacks on AES, NIST P-256, DSA-3072, and RSA-3072 use many standard cryptanalytic techniques cited in Sections 2, 3, 4, and 5. We introduce new cost analyses in all four sections, and new algorithm improvements in Sections 3, 4, and 5; our improvements are critical for beating 2^{128} in Section 5. In Sections 2, 3, and 4 the standard techniques were already adequate to (heuristically) disprove the standard 2^{128} concrete-security conjectures, but as far as we know we were the first to point out these contradictions. We do not think the contradictions were obvious; in many cases the standard techniques were published decades *before* the conjectures!

This paper was triggered by a 23 February 2012 paper [55], in which Koblitz and Menezes objected to the non-constructive nature of Bellare's security proof [7] for NMAC. Bellare's security theorem states a quantitative relationship between the standard-definition-insecurity of NMAC-h and the standard-definition-insecurity of h: the *existence* of a fast attack on NMAC-h implies the *existence* of a fast attack on h. The objection is that the proof does not reveal a fast method to compute the second attack from the first: the proof left open the possibility that the fastest algorithm that can be *found* to attack NMAC-h is much faster than the fastest algorithm that can be *found* to attack h.

An early-March update of [55] added weight to this objection by pointing out the (heuristic) existence of a never-to-be-found fast algorithm to attack any 128-bit function h. The success probability of the algorithm was only about 2^{-64}, but this was still enough to disprove Bellare's security conjectures. Koblitz and Menezes commented on "how difficult it is to appreciate all the security implications of assuming that a function has prf-security even against unconstructible adversaries".

Compared to [55], we analyze a much wider range of attacks, including higher-probability PRF attacks and attacks against various public-key systems, showing that the difficulties here go far beyond PRF security. We also show quantitative variations of the difficulties between one algorithm cost metric and another, and we raise the possibility of eliminating the difficulties by carefully selecting a cost metric.

Readers who find these topics interesting may also be interested in the followup paper [56] by Koblitz and Menezes, especially the detailed discussion in [56, Section 2] of "two examples where the non-uniform model led researchers astray". See also Appendices Q.13, Q.14, and Q.15 of our paper for further comments on the concept of non-uniformity.

2 Breaking AES

This section analyzes the cost of various attacks against AES. All of the attacks readily generalize to other block ciphers; none of the attacks exploit any particular weakness of AES. We focus on AES because of its relevance in practice and to have concrete numbers to illustrate the attacks.

All of the (single-target) attacks here are "PRP" attacks: i.e., attacks that distinguish the cipher outputs for a uniform random key (on attacker-selected inputs) from outputs of a uniform random permutation. Some of the attacks go further, recovering the cipher key, but this is not a requirement for a distinguishing attack.

2.1. Breaking AES with MD5. We begin with an attack that does not use any precomputations. This attack is feasible, and in fact quite efficient; its success probability is low, but not nearly as low as one might initially expect. This is a warmup for the higher-success-probability attack of Section 2.2.

Let P be a uniform random permutation of the set $\{0,1\}^{128}$; we label elements of this set in little-endian form as integers $0,1,2,\ldots$ without further comment. The pair $(P(0),P(1))$ is nearly a uniform random 256-bit string: it avoids 2^{128} strings of the form (x,x) but is uniformly distributed among the remaining $2^{256}-2^{128}$ strings.

If k is a uniform random 128-bit string then the pair $(\mathrm{AES}_k(0),\mathrm{AES}_k(1))$ is a highly nonuniform random 256-bit string, obviously incapable of covering more than 2^{128} possibilities. One can reasonably guess that an easy way to distinguish this string from $(P(0),P(1))$ is to feed it through MD5 and output the first bit of the result. The success probability p of this attack — the absolute difference between the attack's average output for input $(\mathrm{AES}_k(0),\mathrm{AES}_k(1))$ and the attack's average output for input $(P(0),P(1))$ — is far below 1, but it is almost certainly above 2^{-80}, and therefore many orders of magnitude above 2^{-128}. See Appendix V for relevant computer experiments.

To understand why p is so large, imagine replacing the first bit of MD5 with a uniform random function from $\{0,1\}^{256}$ to $\{0,1\}$, and assume for simplicity that the 2^{128} keys k produce 2^{128} distinct strings $(\mathrm{AES}_k(0),\mathrm{AES}_k(1))$. Each key k then has a 50% chance of choosing 0 and a 50% chance of choosing 1, and these choices are independent, so the probability that $2^{127}+\delta$ keys k choose 1 is exactly $\binom{2^{128}}{2^{127}+\delta}/2^{2^{128}}$; the probability that *at least* $2^{127}+\delta$ keys k choose 1 is exactly $\sum_{i\ge\delta}\binom{2^{128}}{2^{127}+i}/2^{2^{128}}$; the probability that *at most* $2^{127}-\delta$ keys k choose 1 is the same. The other $2^{256}-2^{129}$ possibilities for $(P(0),P(1))$ are practically guaranteed to have far smaller bias. Consequently p is at least $\approx\delta/2^{128}$ with probability approximately $2\sum_{i\ge\delta}\binom{2^{128}}{2^{127}+i}/2^{2^{128}} \approx 1-\mathrm{erf}(\delta/\sqrt{2^{127}}) \approx \exp(-\delta^2/2^{127})$, where erf is the standard error function. For example, p is at least $\approx 2^{-65}$ with probability above 30%, and is at least $\approx 2^{-80}$ with probability above 99.997%.

Of course, MD5 is not actually a uniform random function, but it would be astonishing for MD5 to interact with AES in such a way as to spoil this attack. More likely is that there are some collisions in $k\mapsto(\mathrm{AES}_k(0),\mathrm{AES}_k(1))$; but such collisions are rare unless AES is deeply flawed, and in any event will tend to push δ away from 0, helping the attack.

2.2. Precomputing larger success probabilities. The same analysis applies to a modified attack D_s that appends a short string s to the AES outputs $(\mathrm{AES}_k(0),\mathrm{AES}_k(1))$ before hashing them: with probability $\approx\exp(-\delta^2/2^{127})$ the attack D_s has success probability at least $\approx\delta/2^{128}$. If s is long enough to push the hash inputs beyond one block of MD5 input then the iterated structure of MD5 seems likely to spoil the attack, so we define D_s using "capacity-1024 Keccak" rather than MD5.

Consider, for example, $\delta=2^{67}$: with probability $\approx 1-\mathrm{erf}(2^{3.5})\approx 2^{-189}$ the attack D_s has success probability at least $\approx 2^{-61}$. There are 2^{192} choices of 192-bit strings s, so presumably at least one of them will have D_s having success probability at least $\approx 2^{-61}$. Of course, actually *finding* such an s would require inconceivable amounts of computation by the best methods known (searching 2^{189} choices of s, and computing 2^{128} hashes for each choice); but this is not relevant to the definition of insecurity, which considers only the time taken by D_s.

More generally, for any $n \in \{0, 1, 2, \ldots, 64\}$ and any s, with probability $\approx 1 - \mathrm{erf}(2^{n+0.5}) \approx \exp(-2^{2n+1})$, the attack D_s has success probability at least $\approx 2^{n-64}$. There are $2^{3 \cdot 2^{2n}}$ choices of $(3 \cdot 2^{2n})$-bit strings s, and $2^{3 \cdot 2^{2n}}$ is considerably larger than $\exp(2^{2n+1})$, so presumably at least one of these values of s will have D_s having success probability at least $\approx 2^{n-64}$.

Similar comments apply to essentially any short-key cipher. There almost certainly *exists* a $(3 \cdot 2^{2n})$-bit string s such that the following simple attack achieves success probability $\approx 2^{n-K/2}$, where K is the number of bits in the cipher key: query $2K$ bits of cipher output, append s, and hash the result to 1 bit. Later we will write p for the success probability; note that the string length is close to $2^K p^2$.

As n increases, the cost of hashing $3 \cdot 2^{2n} + 2K$ bits grows almost linearly with 2^{2n} in the RAM metric and the NAND metric. It grows more quickly in the AT metric: storing the $3 \cdot 2^{2n}$ bits of s uses area at least $3 \cdot 2^{2n}$, and even a heavily parallelizable hash function will take time proportional to 2^n simply to communicate across this area, for a total cost proportional to 2^{3n}. In each metric there are also lower-order terms reflecting the cost of hashing per bit; we suppress these lower-order terms since our concern is with much larger gaps.

2.3. Iteration (Hellman etc.). Large success probabilities are more efficiently achieved by a different type of attack that iterates, e.g., the function $f_7 : \{0,1\}^{128} \to \{0,1\}^{128}$ defined by $f_7(k) = \mathrm{AES}_k(0) \oplus 7$.

Choose an attack parameter n. Starting from $f_7(k)$, compute the sequence of iterates $f_7(k), f_7^2(k), f_7^3(k), \ldots, f_7^{2^n}(k)$. Look up each of these iterates in a table containing the precomputed quantities $f_7^{2^n}(0), f_7^{2^n}(1), \ldots, f_7^{2^n}(2^n - 1)$. If $f_7^j(k)$ matches $f_7^{2^n}(i)$, recompute $f_7^{2^n - j}(i)$ as a guess for k, and verify this guess by checking $\mathrm{AES}_k(1)$.

This computation finds the target key k if k matches any of the following keys: $0, f_7(0), \ldots, f_7^{2^n-1}(0)$; $1, f_7(1), \ldots, f_7^{2^n-1}(1)$; etc. If n is not too large (see the next paragraph) then there are close to 2^{2n} different keys here. The computation involves $\le 2^n$ initial iterations; 2^n table lookups; and, in case of a match, $\le 2^n$ iterations to recompute $f_7^{2^n-j}(i)$. The *precomputation* performs many more iterations, but this precomputation is only the cost of *finding* the algorithm, not the cost of *running* the algorithm.

This heuristic analysis begins to break down as $3n$ approaches the key size K. The central problem is that a chain $f_7(i), f_7^2(i), \ldots$ could collide with one of the other $2^n - 1$ chains; this occurs with probability $\approx 2^{3n}/2^K$, since there are 2^n keys in this chain and almost 2^{2n} keys in the other chains. The colliding chains will then merge, reducing the coverage of keys and at the same time requiring extra iterations to check more than one value of i. This phenomenon loses a small constant factor in the algorithm performance for $n \approx K/3$ and much more for larger n.

Assume from now on that n is chosen to be close to $K/3$. The algorithm then has success chance $\approx 2^{-K/3}$. The algorithm cost is on the scale of $2^{K/3}$ in both the RAM metric and the NAND metric; for the NAND metric one computes the 2^n independent table lookups by sorting and merging.

This attack might not sound better (in the RAM metric) than the earlier attack D_s, which achieves success chance $\approx 2^{-K/3}$ for some string s with $\approx 2^{K/3}$ bits. The critical feature of this attack is that it recognizes its successes. If the attack fails to find k then one can change 7 to another number and try again, almost doubling the success chance of the algorithm at the expense of doubling its cost; for comparison, doubling the success chance of D_s requires quadrupling its cost. Repeating this attack $2^{K/3}$ times reaches success chance ≈ 1 at cost $2^{2K/3}$.

In the *AT* metric this attack is much more expensive. The table of precomputed quantities $f_7^{2^n}(0), f_7^{2^n}(1), \ldots, f_7^{2^n}(2^n-1)$ uses area on the scale of 2^n, and computing $f_7^{2^n}(k)$ takes time on the scale of 2^n, for a total cost on the scale of 2^{2n} for an attack that finds $\approx 2^{2n}$ keys. One can *compute* $f_7^{2^n}(0), f_7^{2^n}(1), \ldots, f_7^{2^n}(2^n-1)$ in parallel within essentially the same bounds on time and area, replacing each precomputed key with a small circuit that computes the key from scratch; precomputation does not change the exponent of the attack. One can, more straightforwardly, compute any reasonable sequence of 2^{2n} guesses for k within essentially the same cost bound. Achieving success probability p costs essentially $2^K p$.

2.4. Multiple targets. Iteration becomes more efficient when there are multiple targets: U cipher outputs $\mathrm{AES}_{k_1}(0), \mathrm{AES}_{k_2}(0), \ldots, \mathrm{AES}_{k_U}(0)$ for U independent uniform random keys $k_1, \ldots, k_U$. Assume for simplicity that U is much smaller than 2^K; the hypothesis $U \le 2^{K/4}$ suffices for all heuristics used below.

Compute the iterates $f_7(k_1), f_7^2(k_1), \ldots, f_7^{2^n}(k_1)$, and similarly for each of $k_2, \ldots, k_U$; this takes $2^n U$ iterations. Look up each iterate in a table of $2^n U$ precomputed keys. Handle any match as above.

In the RAM metric or the NAND metric this attack has cost on the scale of $2^n U$, just like applying the previous attack to the U keys separately. The benefit of this attack is that it uses a larger table, producing a larger success probability for each key: the precomputation covers $2^{2n}U$ keys instead of just 2^{2n} keys. To avoid excessive chain collisions one must limit 2^n to $2^{K/3}U^{-1/3}$ so that $2^{3n}U$ does not grow past 2^K; the attack then finds each key with probability $2^{2n}U/2^K = 2^{-K/3}U^{1/3}$, with a cost of $2^n = 2^{K/3}U^{-1/3}$ per key, a factor of $U^{2/3}$ better than handling each key separately. Finding each key with high probability costs $2^{2K/3}U^{-2/3}$ per key.

As before, the *AT* metric assigns a much larger cost than the RAM and NAND metrics. The computation of $f_7^{2^n}(k_1), f_7^{2^n}(k_2), \ldots, f_7^{2^n}(k_U)$ is trivially parallelized, taking time on the scale of 2^n, but the $2^n U$ precomputed keys occupy area $2^n U$, for a total cost on the scale of $2^{2n}U$, i.e., 2^{2n} per key, for success probability $2^{2n}U/2^K$ per key. Note that one can carry out the precomputation using essentially the same area and time. There is a large benefit from handling U keys together — finding all U keys costs essentially 2^K, i.e., $2^K/U$ per key — but this benefit exists whether or not precomputation costs are taken into account.

2.5. Comparison. We summarize the insecurity established by the best attacks presented above. Achieving success probability p against U keys costs

– RAM metric: $\approx 2^K p^2$ for $p \le 2^{-K/3} U^{-2/3}$; $\approx (2^{2K/3} U^{-2/3}) p$ for larger p.
– NAND metric: same.
– AT metric: $\approx 2^{3K/2} p^3$ for $p \le 2^{-K/4} U^{-1/2}$; $\approx 2^K U^{-1} p$ for larger p.

Figure G.1 graphs these approximations for $U = 1$, along with the cost of exhaustive search.

2.6. Previous work. All of the attacks described here have appeared before. In fact, when the conjectures in [16, Section 3.6] and [7, Section 3.2] were made, they were already inconsistent with known attacks.

The iteration idea was introduced by Hellman in [45] for the special case $U = 1$. Many subsequent papers (see, e.g., [25] and [49]) have explored variants and refinements of Hellman's attack, including the easy generalization to larger U. Hellman's goal was to attack many keys for a lower RAM cost than attacking each key separately; Hellman advertised a "cost per solution" of $2^{2K/3}$ using a precomputed table of size $2^{2K/3}$. The generalization to larger U achieves the same goal at lower cost, but the special case $U = 1$ remains of interest as a non-uniform single-key attack.

Koblitz and Menezes in [55] recently considered a family of attacks analogous to D_s. They explained that there should be a short string s where D_s has success probability at least $\approx 2^{-K/2}$, and analyzed some consequences for provable concrete secret-key security. However, they did not analyze higher levels of insecurity.

Replacing D_s with a more structured family of attacks, namely linear cryptanalysis, can be *proven* to achieve insecurity $2^{-K/2}$ at low cost. (See, for example, [39, Section 7], which says that this is "well known in complexity theory".) De, Trevisan, and Tulsiani in [36] proved cost $\approx 2^K p^2$, for both the RAM metric and the NAND metric, for any insecurity level p. A lucid discussion of the gap between these attacks and exhaustive search appears in [36, Section 1], but without any analysis of the resulting trouble for the literature on provable concrete secret-key security, and without any analysis of possible fixes.

Biham, Goren, and Ishai in [23, Section 1.1] pointed out that Hellman's attack causes problems for defining strong one-way functions. The only solution that they proposed was adding uniformity. Note that this solution abandons the goal of giving a definition for, e.g., the strength of AES as a one-way function, or the strength of protocols built on top of AES. We analyze this solution in detail in Appendix B.5.

Our AT analysis appears to be new. In particular, we are not aware of previous literature concluding that switching to the AT metric removes essentially all of the benefit of precomputation for large p, specifically $p > 2^{-K/4} U^{-1/2}$.

3 Breaking the NIST P-256 elliptic curve

This section analyzes the cost of an attack against NIST P-256 [67], an elliptic curve of 256-bit prime order ℓ over a 256-bit prime field $\mathbf{F}_p$. The attack computes discrete logarithms on this curve, recovering the secret key from the public key and thus completely breaking typical protocols that use NIST P-256.

The attack does not exploit any particular weakness of NIST P-256. Switching from NIST P-256 to another group of the same size (another curve over the same field, a curve over another field, a hyperelliptic curve, a torus, etc.) does not stop the attack. We focus on NIST P-256 for both concreteness and practical relevance, as in the previous section.

3.1. The standard attack without precomputation. Let P be the specified base point on the NIST P-256 curve. The discrete-logarithm problem on this curve is to find, given another point Q on this curve, the unique integer k modulo ℓ such that $Q = kP$. The standard attack against the discrete-logarithm problem is the parallelization by van Oorschot and Wiener [72] of Pollard's rho method [73], described in the following paragraphs.

This attack uses a pseudorandom walk on the curve points. To obtain the $(i+1)$-st point P_{i+1}, apply a hash function $h : \mathbf{F}_p \to I$ to the x-coordinate of P_i, select a step $S_{h(x(P_i))}$ from a sequence of precomputed steps $S_j = r_j P$ (with random scalars r_j for $j \in I$), and compute $P_{i+1} = P_i + S_{h(x(P_i))}$. The size of I is chosen large enough to have the walk simulate a uniform random walk; a common choice, recommended in [87], is $|I| = 20$. The walk continues until it hits a distinguished point: a point P_i where the last t bits of $x(P_i)$ are equal to zero. Here t is an attack parameter.

The starting point of the bth walk is of the form $aP + bQ$ where a is chosen randomly. Each step increases the multiple of P, so the distinguished point has the form $a'P + bQ$ for known a', b. The triple $(a'P + bQ, a', b)$ is stored and a new walk is started from a different starting point. If two walks hit the same distinguished point then $a'P + bQ = c'P + dQ$ which gives $(a' - c')P = (d - b)Q$; by construction $d \not\equiv b \bmod \ell$, revealing $k \equiv (a' - c')/(d - b) \bmod \ell$.

After $\sqrt{\ell} \approx 2^{128}$ additions (in approximately 2^{128-t} walks, using storage 2^{128-t}), there is a high chance that the same point has been obtained in two different walks. This collision is recognized from a repeated distinguished point within approximately 2^t additional steps.

3.2. Precomputed distinguished points. To use precomputations in this attack, build a database of triples of the form $(a'P, a', 0)$, i.e., starting each walk at a multiple of P. The attack algorithm takes this database and starts a new walk at $aP + bQ$ for random a and b. If this walk ends in a distinguished point present in the database, the DLP is solved. If the walk continues for more than 2^{t+1} steps (perhaps because it is in a cycle) or reaches a distinguished point not present in the database, the attack starts again from a new pair (a, b).

The parameter t is critical for RAM cost here, whereas it did not significantly affect RAM cost in Section 3.1. Choose t as $\lceil(\log_2 \ell)/3\rceil$. One can see from the following analysis that significantly smaller values of t are much less effective, and that significantly larger values of t are much more expensive without being much more effective.

Construct the database to have exactly 2^t distinct triples, each obtained from a walk of length at least 2^t, representing a total of at least 2^{2t} (and almost certainly $O(2^{2t})$) points. Achieving this requires searching for starting points in the precomputation (and optionally also varying the steps S_j and the hash function) as follows. A point that enters a cycle without reaching a distinguished point is discarded. A point that reaches a distinguished point in fewer than 2^t steps is discarded; each point survives this with probability approximately $(1 - 1/2^t)^{2^t} \approx 1/e$. A point that produces a distinguished point already in the database is discarded; to see that a point survives this with essentially constant probability (independent of ℓ), observe that each new step has chance 2^{-t} of reaching a distinguished point, and chance $O(2^{2t}/\ell) = O(2^{-t})$ of reaching one of the previous $O(2^{2t})$ points represented by the database. Computer experiments that we reported in [**22**], as a followup to this paper, show that all the O constants here are reasonably close to 1.

Now consider a walk starting from $aP + bQ$. This walk has chance approximately $1/e$ of continuing for at least 2^t steps. If this occurs then those 2^t steps have chance approximately $1-(1-2^{2t}/\ell)^{2^t} \approx 1-\exp(-2^{3t}/\ell) \geq 1-1/e$ of reaching one of the 2^{2t} points in the precomputed walks that were within 2^t of the distinguished points in the database. If this occurs then the walk is guaranteed to reach a distinguished point in the database within a total of 2^{t+1} steps. The algorithm thus succeeds (in this way) with probability at least $(1-1/e)/e \approx 0.23$. This is actually an underestimate, since the algorithm can also succeed with an early distinguished point or a late collision.

To summarize, the attack uses a database of approximately $\sqrt[3]{\ell}$ distinguished points; one run of the attack uses approximately $2\sqrt[3]{\ell}$ curve additions and succeeds with considerable probability. The overall attack cost in the RAM metric is a small constant times $\sqrt[3]{\ell}$. The security of NIST P-256 in this metric has thus dropped to approximately 2^{86}. Note that the precomputation here is on the scale of 2^{170}, much larger than the precomputation in Section 2.3 but much smaller than the precomputation in Section 2.2.

In the NAND metric it is simplest to run each walk for exactly 2^{t+1} steps, keeping track of the first distinguished point found by that walk and then comparing that distinguished point to the 2^t points in the database. The overall attack cost is still on the scale of $\sqrt[3]{\ell}$.

In the AT metric the attack cost is proportional to $\sqrt[3]{\ell}^2$, larger than the standard $\sqrt{\ell}$. In this metric one does better by running many walks in parallel: if Z points are precomputed, one should run approximately Z walks in parallel with inputs depending on Q. The precomputation then covers $2^t Z$ points, and the computations involving Q cover approximately $2^t Z$ points, leading to a high probability of success when $2^t Z$ reaches $\sqrt{\ell}$. The AT cost is also $2^t Z$. This attack has the same cost as the standard Pollard rho method, except for small constants; there is no benefit in the precomputations.

3.3. Comparison. We summarize the insecurity established by the best attacks presented above. Achieving success probability p costs

- RAM metric: $\approx(p\ell)^{1/3}$.
- NAND metric: same.
- AT metric: $\approx(p\ell)^{1/2}$.

Figure G.2 graphs these approximations.

3.4. Related work. Kuhn and Struik in [58] and Hitchcock, Montague, Carter, and Dawson in [46] considered the problem of solving multiple DLPs at once. They obtain a speedup of $\sqrt{U}$ per DLP for solving U DLPs at once. Their algorithm reuses the distinguished points found in the attack on Q_1 to attack Q_2, reuses the distinguished points found for Q_1 and Q_2 to attack Q_3, etc. However, their results do not seem to imply our $\sqrt[3]{\ell}$ result: they do not change the average walk length and distinguished-point probabilities, and they explicitly limit U to $c\sqrt[4]{\ell}$ with $c < 1$. See also the recent paper [61] by Lee, Cheon, and Hong, which considered solving DLPs with massive precomputation for trapdoor DL-groups. None of these papers noticed any implications for provable security, and none of them went beyond the RAM metric.

Our followup paper [22] experimentally verified the algorithm stated above, improved it to $1.77 \cdot \sqrt[3]{\ell}$ additions using $\sqrt[3]{\ell}$ distinguished points, extended it to DLPs in intervals (using slightly more additions), and showed constructive applications in various protocols.

4 Breaking DSA-3072

This section briefly analyzes the cost of an attack against the DSA-3072 signature system. The attack computes discrete logarithms in the DSA-3072 group, completely breaking the signature system.

DSA uses the unique order-q subgroup of the multiplicative group $\mathbf{F}_p^*$, where p and q are primes with q (and not q^2) dividing $p-1$. DSA-3072 uses a 3072-bit prime p and is claimed to achieve 2^{128} security. The standard parameter choices for DSA-3072 specify a 256-bit prime q, allowing the 2^{86} attack explained in Section 3, but this section assumes that the user has stopped this attack by increasing q to 384 bits (at a performance penalty).

4.1. The attack. Take $y = 2^{110}$, and precompute $\log_g x^{(p-1)/q}$ for every prime number $x \le y$, where g is the specified subgroup generator. There are almost exactly $y/\log y \approx 2^{103.75}$ such primes, and each $\log_g x^{(p-1)/q}$ fits into 48 bytes, for a total of $2^{109.33}$ bytes.

To compute $\log_g h$, first try to write h as a quotient h_1/h_2 in $\mathbf{F}_p^*$ with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$, $h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$, and $\gcd\{h_1, h_2\} = 1$; and then try to factor h_1, h_2 into primes $\le y$. If this succeeds then $\log_g h^{(p-1)/q}$ is a known combination of known quantities $\log_g x^{(p-1)/q}$, revealing $\log_g h$. If this fails, try again with hg, hg^2, etc.

One can write h as h_1/h_2 with high probability, approximately $(6/\pi^2)2^{3071}/p$, since there are approximately $(6/\pi^2)2^{3071}$ pairs (h_1, h_2) and two distinct such pairs have distinct quotients. Finding the decomposition of h as h_1/h_2 is a very fast extended-Euclid computation.

The probability that h_1 is y-smooth (i.e., has no prime divisors larger than y) is very close to $u^{-u} \approx 2^{-53.06}$ where $u = 1535/110$. The same is true for h_2; overall the attack requires between $2^{107.85}$ and $2^{108.85}$ iterations, depending on $2^{3071}/p$. Batch trial division, analyzed in detail in Section 5, finds the y-smooth values among many choices of h_1 at very low cost in both the RAM metric and the NAND metric. This attack is much slower in the AT metric.

4.2. Previous work. Standard attacks against DSA-3072 do not rely on precomputation and cost more than 2^{128} in the RAM metric. These attacks have two stages: the first stage computes discrete logarithms of all primes $\leq y$, and the second stage computes $\log_g h$. Normally y is chosen to minimize the cost of the first stage, whereas we replace the first stage by precomputation and choose y to minimize the cost of the second stage.

The simple algorithm reviewed here is not the state-of-the-art algorithm for the second stage; see, e.g., the "special-q descent" algorithms in [51] and [32]. The gap between known algorithms and existing algorithms is thus even larger than indicated in this section. We expect that reoptimizing these algorithms to minimize the cost of the second stage will produce even better results. We emphasize, however, that none of the algorithms perform well in the AT metric.

5 Breaking RSA-3072

This section analyzes the cost of an attack against RSA-3072. The attack completely breaks RSA-3072, factoring any given 3072-bit public key into its prime factors, so it also breaks protocols such as RSA-3072-FDH and RSA-3072-OAEP.

This section begins by stating a generalization of the attack to any RSA key size, and analyzing the asymptotic cost exponents of the generalized attack. It then analyzes the cost more precisely for 3072-bit keys.

5.1. NFS with precomputation. This attack is a variant of NFS, the standard attack against RSA. For simplicity this description omits several NFS optimizations. See [30] for an introduction to NFS.

The attack is determined by four parameters: a "polynomial degree" d; a "radix" m; a "height bound" H; and a "smoothness bound" y. Each of these parameters is a positive integer. The attack also includes a precomputed "factory"

$$F = \left\{ (a,b) \in \mathbf{Z} \times \mathbf{Z} : \begin{matrix} -H \leq a \leq H;\ 0 < b \leq H; \\ \gcd\{a,b\} = 1;\ \text{and } a - bm \text{ is } y\text{-smooth} \end{matrix} \right\}.$$

The standard estimate (see [30]) is that F has $(12/\pi^2)H^2/u^u$ elements where $u = (\log Hm)/\log y$. This estimate combines three approximations: first, there are about $12H^2/\pi^2$ pairs $(a,b) \in \mathbf{Z} \times \mathbf{Z}$ such that $-H \leq a \leq H$, $0 < b \leq H$, and $\gcd\{a,b\} = 1$; second, $a - bm$ has approximately the same smoothness chance as a uniform random integer in $[1, Hm]$; third, the latter chance is approximately $1/u^u$.

The integers N factored by the attack will be between m^d and m^{d+1}. For example, with parameters $m = 2^{256}$, $d = 7$, $H = 2^{55}$, and $y = 2^{50}$, the attack factors integers between 2^{1792} and 2^{2048}. Parameter selection is analyzed later in more detail. The following three paragraphs explain how the attack handles N.

Write N in radix m: i.e., find $n_0, n_1, \ldots, n_d \in \{0, 1, \ldots, m-1\}$ such that $N = n_d m^d + n_{d-1} m^{d-1} + \cdots + n_0$. Compute the "set of relations"

$$R = \left\{(a, b) \in F : n_d a^d + n_{d-1} a^{d-1} b + \cdots + n_0 b^d \text{ is } y\text{-smooth}\right\}$$

using Bernstein's batch trial-division algorithm [19]. The standard estimate is that R has $(12/\pi^2)H^2/(u^u v^v)$ elements where $v = (\log((d+1)H^d m))/\log y$.

We pause the attack description to emphasize two important ways that this attack differs from conventional NFS: first, conventional NFS chooses m as a function of N, while this attack does not; second, conventional NFS computes R by sieving all pairs (a, b) with $-H \le a \le H$ and $0 < b \le H$ to detect smoothness of $a - bm$ and $n_d a^d + \cdots + n_0 b^d$ simultaneously, while this attack computes R by batch trial division of $n_d a^d + \cdots + n_0 b^d$ for the limited set of pairs $(a, b) \in F$.

The rest of the attack proceeds in the same way as conventional NFS. There is a standard construction of a sparse vector modulo 2 for each $(a, b) \in R$, and there is a standard way to convert several linear dependencies between the vectors into several congruences of squares modulo N, producing the complete prime factorization of N; see [30] for details. The number of components of each vector is approximately $2y/\log y$, and standard sparse-matrix techniques find linear dependencies using about $4y/\log y$ simple operations on dense vectors of length $2y/\log y$. If the number of elements of R is larger than the number of components of each vector then linear dependencies are guaranteed to exist.

5.2. Asymptotic exponents. Write $L = \exp((\log N)^{1/3}(\log\log N)^{2/3})$. For the RAM metric it is best to choose

$$\begin{aligned}
d &\in (1.1047\ldots + o(1))(\log N)^{1/3}(\log\log N)^{-1/3},\\
\log m &\in (0.9051\ldots + o(1))(\log N)^{2/3}(\log\log N)^{1/3},\\
\log y &\in (0.8193\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (0.8193\ldots + o(1))\log L,\\
\log H &\in (1.0034\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (1.0034\ldots + o(1))\log L.
\end{aligned}$$

so that

$$\begin{aligned}
u &\in (1.1047\ldots + o(1))(\log N)^{1/3}(\log\log N)^{-1/3},\\
u\log u &\in (0.3682\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (0.3682\ldots + o(1))\log L,\\
d\log H &\in (1.1085\ldots + o(1))(\log N)^{2/3}(\log\log N)^{1/3},\\
v &\in (2.4578\ldots + o(1))(\log N)^{1/3}(\log\log N)^{-1/3},\\
v\log v &\in (0.8193\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (0.8193\ldots + o(1))\log L.
\end{aligned}$$

Out of the $L^{2.0068\ldots+o(1)}$ pairs (a,b) with $-H \le a \le H$ and $0 < b \le H$, there are $L^{1.6385\ldots+o(1)}$ pairs in the factory F, and $L^{0.8193\ldots+o(1)}$ relations in R, just enough to produce linear dependencies if the $o(1)$ terms are chosen appropriately. Linear algebra uses $y^{2+o(1)} = L^{1.6385\ldots+o(1)}$ bit operations.

The total RAM cost of this factorization algorithm is thus $L^{1.6385\ldots+o(1)}$. For comparison, factorization is normally claimed to cost $L^{1.9018\ldots+o(1)}$ (in the RAM metric) with state-of-the-art variants of NFS. Similar comments apply to the NAND metric.

This algorithm runs into trouble in the AT metric. The algorithm needs space to store all the elements of F, and can compute R in time $L^{o(1)}$ using a chip of that size (applying ECM to each input in parallel rather than using batch trial division), but even the most heavily parallelized sparse-matrix techniques need much more than $L^{o(1)}$ time, raising the AT cost of the algorithm far above the size of F. A quantitative analysis shows that one obtains a better cost exponent by skipping the precomputation of F and instead computing the elements of F one by one on a smaller circuit, for AT cost $L^{1.9760\ldots+o(1)}$.

5.3. RAM cost for RSA-3072. This attack breaks RSA-3072 with RAM cost considerably below the 2^{128} security level usually claimed for RSA-3072. Of course, justifying this estimate requires replacing the above $o(1)$ terms with more precise cost analyses.

For concreteness, assume that the RAM supports 128-bit pointers, unit-cost 256-bit vector operations, and unit-cost 256-bit floating-point multiplications. As justification for these assumptions, observe that real computers ten years ago supported 32-bit pointers, unit-cost 64-bit vector operations, and unit-cost 64-bit floating-point multiplications; that the RAM model requires operations to scale logarithmically with the machine size; and that previous NFS cost analyses implicitly make similar assumptions.

Take $m = 2^{384}$, $d = 7$, $H = 2^{62}+2^{61}+2^{57}$, and $y = 2^{66}+2^{65}$. There are about $12H^2/\pi^2 \approx 2^{125.51}$ pairs (a,b) with $-H \le a \le H$, $0 < b \le H$, and $\gcd\{a,b\} = 1$, and the integers $a - bm$ have smoothness chance approximately $u^{-u} \approx 2^{-18.42}$ where $u = (\log Hm)/\log y \approx 6.707$, so there are about $2^{107.09}$ pairs in the factory F. Each pair in F is small, easily encoded as just 16 bytes.

The quantities $n_d a^d + n_{d-1}a^{d-1}b + \cdots + n_0 b^d$ are bounded by $(d+1)mH^d \approx 2^{825.3}$. If they were uniformly distributed up to this bound then they would have smoothness chance approximately $v^{-v} \approx 2^{-45.01}$ where $v = (\log((d+1)mH^d))/\log y \approx 12.395$, so there would be approximately $(12H^2/\pi^2)u^{-u}v^{-v} \approx 2^{62.08}$ relations, safely above $2y/\log y \approx 2^{62.06}$. The quantities $n_d a^d + n_{d-1}a^{d-1}b + \cdots + n_0 b^d$ are actually biased towards smaller values and thus have larger smoothness chance, but this refinement is unnecessary here.

Batch trial division checks smoothness of 2^{58} of these quantities simultaneously; here 2^{58} is chosen so that the product of those quantities is larger (about $2^{67.69}$ bits) than the product of all the primes $\le y$ (about $2^{67.11}$ bits). The main steps in batch trial division are computing a product tree of these quantities and then computing a scaled remainder tree. Bernstein's cost analysis in [20, Section 3] shows that the overall cost of these two steps, for T inputs having a B-bit product, is approximately $(5/6)\log_2 T$ times the cost of a single multiplication of two $(B/2)$-bit integers. For us $T = 2^{58}$ and $B \approx 2^{67.69}$, and the cost of batch trial division is approximately $2^{5.59}$ times the cost of multiplying two $(B/2)$-bit integers; the total cost of smoothness detection for all $(a, b) \in F$ is approximately $2^{54.68}$ times the cost of multiplying two $(B/2)$-bit integers.

It is easiest to follow a standard floating-point multiplication strategy, dividing each $(B/2)$-bit input into $B/(2w)$ words for some word size $w \in \Omega(\log_2 B)$ and then performing three real floating-point FFTs of length B/w. Each FFT uses approximately $(17/9)(B/w)\log_2(B/w)$ arithmetic operations (additions, subtractions, and multiplications) on words of slightly more than $2w$ bits, for a total of $(17/3)(B/w)\log_2(B/w)$ arithmetic operations. A classic observation of Schönhage [82] is that the RAM metric allows constant-time multiplication of $\Theta(\log_2 B)$-bit integers in this context even if the machine model is not assumed to be equipped with a multiplier, since one can afford to build large multiplication tables; but it is simpler to take advantage of the hypothesized 256-bit multiplier, which comfortably allows $w = 69$ and $B/w < 2^{61} + 2^{60}$, for a total multiplication cost of $2^{70.03}$. Computing R then costs approximately $2^{124.71}$.

Linear algebra involves $2^{63.06}$ simple operations on vectors of length $2^{62.06}$. Each operation produces each output bit by xoring together a small number of input bits, on average fewer than 32 bits. A standard block-Wiedemann computation merges 256 xors of bits into a single 256-bit xor with negligible overhead, for a total linear-algebra cost of $2^{122.12}$. All other steps in the algorithm have negligible cost, so the final factorization cost is $2^{124.93}$.

5.4. Previous work. There are two frequently quoted cost exponents for NFS without precomputation. Buhler, Lenstra, and Pomerance in [30] obtained RAM cost $L^{1.9229\ldots+o(1)}$. Coppersmith in [33] introduced a "multiple number fields" tweak and obtained RAM cost $L^{1.9018\ldots+o(1)}$.

Coppersmith also introduced NFS with precomputation in [33], using ECM for smoothness detection. Coppersmith called his algorithm a "factorization factory", emphasizing the distinction between precomputation time (building the factory) and computation time (running the factory). Coppersmith computed the same RAM exponent $1.6385\ldots$ shown above for the cost of one factorization using the factory.

We save a subexponential factor in the RAM cost of Coppersmith's algorithm by switching from ECM to batch trial division. This is not visible in the asymptotic exponent $1.6385\ldots$ but is important for RSA-3072. Our concrete analysis of RSA-3072 security is new, and as far as we know is the *first concrete analysis of Coppersmith's algorithm.*

Bernstein in [18] obtained AT exponent $1.9760\ldots$ for NFS without precomputation, and emphasized the gap between this exponent and the RAM exponent $1.9018\ldots$. Our AT analysis of NFS with precomputation, and in particular our conclusion that this precomputation increases the AT cost of NFS, appears to be new.

References (see full paper for more references, appendices)

[7] Bellare, M., *New proofs for NMAC and HMAC: security without collision-resistance*, in Crypto 2006 [**40**] (2006), 602–619. Cited in §1, §1.1, §1.3, §2.6.

[11] Bellare, M., Kilian, J., Rogaway, P., *The security of cipher block chaining*, in Crypto 1994 [**38**] (1994), 341–358; see also newer version [**12**]. Cited in §1.2.

[12] Bellare, M., Kilian, J., Rogaway, P., *The security of the cipher block chaining message authentication code*, Journal of Computer and System Sciences **61** (2000), 362–399; see also older version [**11**]. Cited in §1, §1, §1, §1.2, §1.2, §1.2.

[14] Bellare, M., Rogaway, P., *Optimal asymmetric encryption — how to encrypt with RSA*, in Eurocrypt 1994 [**37**] (1995), 92–111. Cited in §1.

[15] Bellare, M., Rogaway, P., *The exact security of digital signatures: how to sign with RSA and Rabin*, in Eurocrypt 1996 [**64**] (1996), 399–416. Cited in §1, §1.1.

[16] Bellare, M., Rogaway, P., *Introduction to modern cryptography*, 2005. URL: `http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html`. Cited in §1, §1.1, §2.6.

[18] Bernstein, D. J., *Circuits for integer factorization: a proposal* (2001). URL: `http://cr.yp.to/papers.html#nfscircuit`. Cited in §5.4.

[19] Bernstein, D. J., *How to find smooth parts of integers* (2004). URL: `http://cr.yp.to/papers.html#smoothparts`. Cited in §5.1.

[20] Bernstein, D. J., *Scaled remainder trees* (2004). URL: `http://cr.yp.to/papers.html#scaledmod`. Cited in §5.3.

[22] Bernstein, D. J., Lange, T., *Computing small discrete logarithms faster*, in Indocrypt 2012 [**41**] (2012), 317–338. Cited in §3.2, §3.4.

[23] Biham, E., Goren, Y. J., Ishai, Y., *Basing weak public-key cryptography on strong one-way functions*, in TCC 2008 [**31**] (2008), 55–72. Cited in §2.6.

[25] Biryukov, A., Shamir, A., *Cryptanalytic time/memory/data tradeoffs for stream ciphers*, in Asiacrypt 2000 [**70**] (2000), 1–13. Cited in §2.6.

[27] Bogdanov, A., Khovratovich, D., Rechberger, C., *Biclique cryptanalysis of the full AES*, in Asiacrypt 2011 [**60**] (2011), 344–371. Cited in §1.

[29] Brent, R. P., Kung, H. T., *The area-time complexity of binary multiplication*, Journal of the ACM **28** (1981), 521–534. Cited in §1.2.

[30] Buhler, J. P., Lenstra, H. W., Jr., Pomerance, C., *Factoring integers with the number field sieve*, in [**63**] (1993), 50–94. Cited in §5.1, §5.1, §5.1, §5.4.

[31] Canetti, R. (editor), *TCC 2008*, LNCS, 4948, Springer, 2008. See [23].

[32] Commeine, A., Semaev, I., *An algorithm to solve the discrete logarithm problem with the number field sieve*, in PKC 2006 [**91**] (2006), 174–190. Cited in §4.2.

[33] Coppersmith, D., *Modifications to the number field sieve*, Journal of Cryptology **6** (1993), 169–180. Cited in §5.4, §5.4.

[35] De, A., Trevisan, L., Tulsiani, M., *Non-uniform attacks against one-way functions and PRGs*, Electronic Colloquium on Computational Complexity **113** (2009); see also newer version [**36**].

[36] De, A., Trevisan, L., Tulsiani, M., *Time space tradeoffs for attacks against one-way functions and PRGs*, in Crypto 2010 [**75**] (2010), 649–665; see also older version [**35**]. Cited in §2.6, §2.6.

[37] De Santis, A. (editor), *Eurocrypt 1994*, LNCS, 950, Springer, 1995. See [14].

[38] Desmedt, Y. (editor), *Crypto 1994*, LNCS, 839, Springer, 1994. See [11].

[39] Dodis, Y., Steinberger, J., *Message authentication codes from unpredictable block ciphers*, in Crypto 2009 [**44**] (2009), 267–285. Cited in §2.6.

[40] Dwork, C. (editor), *Crypto 2006*, LNCS, 4117, Springer, 2006. See [7].

[41] Galbraith, S., Nandi, M. (editors), *Indocrypt 2012*, LNCS, 7668, Springer, 2012. See [22].

[44] Halevi, S. (editor), *Crypto 2009*, LNCS, 5677, Springer, 2009. See [39].

[45] Hellman, M. E., *A cryptanalytic time-memory tradeoff*, IEEE Transactions on Information Theory **26** (1980), 401–406. Cited in §2.6.

[46] Hitchcock, Y., Montague, P., Carter, G., Dawson, E., *The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves*, International Journal of Information Security **3** (2004), 86–98. Cited in §3.4.

[49] Hong, J., Sarkar, P., *New applications of time memory data tradeoffs*, in Asiacrypt 2005 [**78**] (2005), 353–372. Cited in §2.6.

[51] Joux, A., Lercier, R., *Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method*, Mathematics of Computation **72** (2003), 953–967. Cited in §4.2.

[53] Katz, J., Lindell, Y., *Introduction to modern cryptography: principles and protocols*, Chapman & Hall/CRC, 2007. Cited in §1.

[55] Koblitz, N., Menezes, A., *Another look at HMAC* (2012). URL: `http://eprint.iacr.org/2012/074`. Cited in §1.2, §1.3, §1.3, §1.3, §2.6.

[56] Koblitz, N., Menezes, A., *Another look at non-uniformity* (2012). URL: `http://eprint.iacr.org/2012/359`. Cited in §1.2, §1.3, §1.3.

[58] Kuhn, F., Struik, R., *Random walks revisited: extensions of Pollard's rho algorithm for computing multiple discrete logarithms*, in SAC 2001 [**89**] (2001), 212–229. Cited in §3.4.

[60] Lee, D. H., Wang, X. (editors), *Asiacrypt 2011*, LNCS, 7073, Springer, 2011. See [27].

[61] Lee, H. T., Cheon, J. H., Hong, J., *Accelerating ID-based encryption based on trapdoor DL using pre-computation*, 11 Jan 2012 (2012). URL: `http://eprint.iacr.org/2011/187`. Cited in §3.4.

[63] Lenstra, A. K., Lenstra, H. W., Jr. (editors), *The development of the number field sieve*, LNM, 1554, Springer, 1993. See [30].

[64] Maurer, U. M. (editor), *Eurocrypt 1996*, LNCS, 1070, Springer, 1996. See [15].

[66] NIST, *Announcing request for candidate algorithm nominations for the Advanced Encryption Standard (AES)* (1997). URL: `http://www.gpo.gov/fdsys/pkg/FR-1997-09-12/pdf/97-24214.pdf`. Cited in §1.

[67] NIST, *Digital signature standard*, Federal Information Processing Standards Publication 186-2 (2000). URL: `http://csrc.nist.gov`. Cited in §3.

[70] Okamoto, T. (editor), *Asiacrypt 2000*, LNCS, 1976, Springer, 2000. See [25].

[72] van Oorschot, P. C., Wiener, M., *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), 1–28. Cited in §3.1.

[73] Pollard, J. M., *Monte Carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918–924. Cited in §3.1.

[75] Rabin, T. (editor), *Crypto 2010*, LNCS, 6223, Springer, 2010. See [36].

[78] Roy, B. (editor), *Asiacrypt 2005*, LNCS, 3788, Springer, 2005. See [49].

[82] Schönhage, A., *Storage modification machines*, SIAM Journal on Computing **9** (1980), 490–508. Cited in §5.3.

[87] Teske, E., *On random walks for Pollard's rho method*, Mathematics of Computation **70** (2001), 809–825. Cited in §3.1.

[89] Vaudenay, S., Youssef, A. M. (editors), *SAC 2001*, LNCS, 2259, Springer, 2001. See [58].

[91] Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (editors), *PKC 2006*, LNCS, 3958, Springer, 2006. See [32].