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

Daniel J. Bernstein[1] 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. There is a flaw in the standard security definitions used in the literature on provable concrete security. The definitions are frequently conjectured to assign a security level of 2^{128} to AES, the NIST P-256 elliptic curve, DSA-3072, RSA-3072, and various higher-level protocols, but they actually assign a far lower security level to each of these primitives and protocols. This flaw undermines security evaluations and comparisons throughout the literature. This paper analyzes the magnitude of the flaw in detail and discusses several strategies for fixing the definitions.

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

1 Introduction

The conventional wisdom is that the "RSA-PSS" signature system is safer than the "RSA-FDH" signature system against generic-hash attacks (attacks in the "random-oracle model"). This wisdom is backed by the following argument:

- The security of RSA-PSS against these attacks is provably very close to the security of RSA per se, i.e., the difficulty of inverting the RSA permutation.
- The security of RSA-FDH against these attacks is provably not *much* worse than the security of RSA, but it could be *somewhat* worse without contradicting any theorems, and it is certainly not better.

RSA-PSS has essentially the same efficiency as RSA-FDH, making it the obvious choice of signature system for RSA users. This argument was introduced by Bellare and Rogaway in [6, Section 1]; the underlying theorems are the main content of [6].

This is a typical example of a popular research topic called "concrete security". Older security definitions were much less precise, considering only the question of what can be asymptotically broken in polynomial time. "Concrete security" pays attention to polynomially bounded factors, giving it a way to describe and analyze the differences in provable security between RSA-PSS and RSA-FDH, the differences in conjectured security between RSA and ECC, etc.

The study of provable concrete security was initiated by Bellare, Kilian, and Rogaway in [3] for secret-key cryptography (showing that AES-CBC-MAC has similar security to AES), by Bellare and Rogaway in [5] for public-key encryption (showing that RSA-OAEP has similar security to RSA), and by Bellare and Rogaway in [6] for public-key signatures. Modern

This work was supported by the National Science Foundation under grant 1018836, and by the European Commission under Contract ICT-2007-216676 ECRYPT II. Permanent ID of this document: 7e044f2408c599254414615c72b3adbf. Date: 2012.06.04.

security theorems are likely, although not guaranteed, to be stated in the language of concrete security.

1.1. The standard insecurity metric. Of course, one cannot meaningfully state a theorem regarding security without having a definition of security. It is standard in the concrete-security literature to define the insecurity of X — where X is a primitive such as a cipher, or a higher-level protocol such as a signature scheme — as the maximum, over all algorithms A ("attacks") that cost at most C, of the probability that A succeeds in breaking X. This insecurity is a function of the cost limit C. Normally the cost limit is separated into (1) a time limit t and (2) a limit q on the number of oracle queries; the standard argument for this separation is that oracle queries are typically more expensive than computation and can often be structurally limited by the system designer (e.g., "stop signing after 2^{32} messages").

For example, Bellare, Kilian, and Rogaway in [4, Section 1.3] defined the PRP-insecurity of AES ("$\mathbf{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(q,t)$") as the maximum, over all algorithms A that cost at most C, of the chance that A PRP-breaks AES. This chance is, in turn, defined as the distance between

- the probability that A prints 1 given an oracle for AES using a secret key and
- the probability that A prints 1 given an oracle for a uniform random permutation of the set of 128-bit strings.

They gave a similar definition of the PRF-insecurity of m-block AES-CBC-MAC (denoted "$\mathbf{Adv}^{\mathrm{prf}}_{\mathrm{CBC}^m\text{-}\mathrm{AES}}(q,t)$"), using a uniform random function rather than a uniform random permutation. Here is the main theorem of [4]: "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 $t' = t + O(mql)$."

NIST's call for AES submissions [1, 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. It seems obvious that this indistinguishability is exactly what is measured by the PRP-insecurity definition: better distinguishing attacks on AES would produce better lower bounds on the PRP-insecurity of AES; the conjecture that we will never break AES, that our best attacks will always be brute-force attacks, seems to be precisely captured by a conjecture regarding the PRP-insecurity of AES. For example, Bellare and Rogaway in [7, Section 3.6] wrote the following:

> "For example we might conjecture something like:
>
> $$\mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{\mathrm{DES}}(A_{t,q}) \leq c_1 \cdot \frac{t/T_{\mathrm{DES}}}{2^{55}} + c_2 \cdot \frac{q}{2^{40}}$$
>
> ... 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
>
> $$\mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{\mathrm{AES}}(B_{t,q}) \leq c_1 \cdot \frac{t/T_{\mathrm{AES}}}{2^{128}} + c_2 \cdot \frac{q}{2^{128}}."$$

One can quibble that the T_{DES} and T_{AES} factors do not properly account for inner-loop speedups in exhaustive key search (see, e.g., [15]), 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.

The same pattern can be found throughout the literature on provable concrete security. A typical theorem states that the insecurity of a complicated object is bounded in terms of the insecurity of a simpler object. The insecurity of a well-studied primitive such as AES or RSA-1024 is typically conjectured to match the success probability of the best attack known. For example, Bellare and Rogaway in [6, Section 1.4], 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 in [2, Section 3.2], evaluating the concrete security of NMAC-h and HMAC-h, hypothesized that "the best attack against h as a PRF is exhaustive key search".

1.2. Cracks in the standard metric. The main goal of this paper is to highlight a quantitative error built into the standard definition of insecurity. Specifically, we conjecture that the most popular cryptographic primitives and protocols are much less secure according to the standard definition than they are in reality.

Sections 3, 4, 5, and 6 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}. This contradicts the conjectures in [6, Section 1.4], [7, Section 3.6], [2, Section 3.2], etc., and undermines the conclusions regarding concrete security of various protocols.

We nevertheless conjecture that any fast *construction* of such an attack has negligible probability of success. This means that there is a very large gap between the actual insecurity of these primitives and their insecurity according to the standard metrics. The standard metrics count only the cost of running the attack, not the cost of finding the attack in the first place.

We do not claim that this gap is consistent across primitives. On the contrary: we identify different gaps for different primitives, and we expect that analyzing more primitives and protocols in the same way will show even more diversity. 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. The standard definition of insecurity appears to be blind to many attacks that are important for actual insecurity, and as far as we can tell does not have much value in predicting actual insecurity.

The error in the standard insecurity metric has a troubling consequence: almost all of the "protocol P_1 is safer than protocol P_2" conclusions in the concrete-security literature could be backwards. The underlying theorems state that standard-definition insecurity of P_1 is smaller than standard-definition insecurity of P_2, but the difference is almost always quantitatively smaller than the scale of gaps that we discuss between standard-definition insecurity of P_i and *actual* insecurity of P_i. It is therefore entirely possible that the actual insecurity of P_1 is larger than the actual insecurity of P_2.

All of the gaps that we discuss are polynomially bounded, so this trouble does not occur for the most extreme comparisons, theorems showing that P_1 is superpolynomially safer than P_2. However, those comparisons never required the precision of "concrete security": they could have been carried out using simpler analyses of polynomially bounded reductions. One can justify studying security with precision ϵ in a model that is arguably within ϵ of reality, but this argument is no longer plausible for most of the literature on provable concrete security.

In Appendix B we discuss several approaches to addressing these difficulties. Some of these approaches rely on switching from "time" to another cost metric for algorithms, although this

creates other problems discussed in Appendix B. We already introduce three different cost metrics in Section 2, and we analyze the impact of the cost metrics upon various attacks in Sections 3, 4, 5, and 6.

1.3. Previous work. For the problem of finding hash-function collisions it is very well known that "the best algorithm that exists" is not a reasonable model of "the best algorithm that can be found". For example, there is a fast algorithm that outputs collisions in SHA-512, but actually finding such an algorithm seems hopeless.

However, hash-function collisions seem to be viewed as an exceptional case. The same model is widely viewed as reasonable for AES security, RSA security, etc.; consider again the conjectures from [6, Section 1.4], [7, Section 3.6], [2, Section 3.2], etc.

The attacks on AES, ECC, DSA, and RSA presented here use standard cryptanalytic techniques published decades ago. Optimizing the asymptotic exponents of these techniques to take advantage of free precomputation is reasonably easy and in some cases has been done before. See Sections 3, 4, 5, and 6 for further references. However, it does not seem to have been pointed out before that these attacks create foundational difficulties for concrete security, undermining the standard insecurity metrics used in hundreds, perhaps thousands, of papers.

This paper was triggered by a recent paper [29], in which Koblitz and Menezes objected to the non-constructive nature of Bellare's security proof [2] for NMAC. The 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. Subsequent updates of [29] 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 with a success probability far above 2^{-128}, and 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 [29], 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.

2 Cost metrics for algorithms

Recall from Section 1 that the insecurity of X is a function mapping a cost limit C to the maximum, over all algorithms A costing at most C, of the probability that A breaks X.

This function depends implicitly on how the "cost" of an algorithm is defined. This section reviews three cost metrics: the RAM metric used in [4]; the NAND metric, an "alternative" mentioned in [4]; and the AT metric defined in [16].

2.1. The RAM metric. Bellare, Kilian, and Rogaway in [4, Section 2.2] fix "some particular Random Access Machine (RAM)" as a model of computation. They define the running time of an algorithm A as "A's actual execution time plus the length of A's description".

There are well-known difficulties in giving a reasonable definition of "execution time" for RAM programs. However, standard workarounds (see, e.g., [14]) limit these difficulties to a much smaller scale than the gaps considered in this paper, so we suppress discussion of the details. We make an exception in Section 6, where the gap is relatively small.

Bellare, Kilian, and Rogaway say that adding the length of the algorithm "eliminates pathologies caused if one can embed arbitrarily large lookup tables in A's description". The obvious example is an algorithm that includes a giant sorted table of pairs $(\mathrm{AES}_k(0), k)$ for all 2^{128} AES keys k, and simply looks up $\mathrm{AES}_k(0)$ in the table to find k; the RAM metric forces this algorithm to pay for the length of the table, not merely the time taken for the table lookup.

The more advanced attacks discussed later in this paper can be viewed as similar "pathologies" that, contrary to the claim in [4], are *not* eliminated by merely adding the length of the algorithm. This view raises the question of whether further changes to the cost metric could stop those attacks.

2.2. The NAND metric. Bellare, Kilian, and Rogaway also consider an "alternative" definition of an algorithm as a circuit "over some fixed basis of gates, like 2-input NAND gates". The cost of an algorithm then "simply means the circuit size".

Counting NAND gates is refreshingly precise and easy to define. Readers might wonder why this NAND metric is an "alternative" rather than the standard definition of algorithm cost. The only answer given in [4] is that the NAND metric is "rather less intuitive" than the RAM metric.

We emphasize that the NAND metric often assigns far larger costs to algorithms than the RAM metric does. In some cases an algorithm taking time T in the RAM metric costs more than T^2 NAND gates. The most important difference is in the cost of random access to a large table, a very fast operation in the RAM metric but a very slow operation in the NAND metric. A batch of n independent random accesses to the same table of size n has similar cost in both metrics (since it can be simulated by sorting), but many algorithms require serial random accesses.

This difference can cause trouble: there are many theorems regarding "time" that are true for the RAM metric but unproven, and presumably false, for the NAND metric. This trouble is discussed further in Appendix B.2. However, for the same reason, one can hope that any "pathologies" in the RAM metric are fixed by the NAND metric. This hope is analyzed in Sections 3, 4, 5, and 6.

2.3. The AT metric. In hardware design it is common to model computation in a completely different way. Computation is performed by a chip, i.e., a rectangular mesh of transistors connected by wires, with at most a few layers of wires at each point in the mesh. Transistors and wires all operate in parallel. It is not difficult to give a formal definition of this model of computation; see, e.g., [16]. This definition has the virtue of being obviously quite close to physical reality.

Our third cost metric for algorithms is the price-performance ratio of a chip: i.e., the product AT of the area A of the chip and the time T taken by the chip. Hardware designers often consider more general functions of (A, T), but the classic product AT remains the default choice of cost metric because it preserves the following two forms of linearity: performing n time-T computations in serial on one area-A chip costs n times as much as performing 1 computation; performing n time-T computations in parallel on n area-A chips (formally, one area-nA chip) also costs n times as much as performing 1 computation.

Brent and Kung showed in [16] that n-bit multiplication costs $n^{1.5+o(1)}$ in the AT metric; for comparison, n-bit multiplication costs only $n^{1+o(1)}$ in the RAM metric and in the NAND metric. Similar comments apply to sorting and to various other high-communication computations. We consider the AT metric in subsequent sections for the same reason that we consider the NAND metric: it is a source of trouble but also a possible solution to "pathologies".

3 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.

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.

3.1. Breaking AES with MD5. Let P be a uniform random permutation of the set $\{0,1\}^{128}$. 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. One can reasonably guess that an easy way to distinguish this from $(P(0),P(1))$ is to feed it through MD5 and inspect the first bit of the result. The success probability of this attack is far below 1, but it is almost certainly above 2^{-80}, and therefore many orders of magnitude above 2^{-128}.

To understand why this works, 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 $\sum_{i\geq\delta}\binom{2^{128}}{2^{127}+i}/2^{2^{128}}$; 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. This attack thus demonstrates insecurity $\approx\delta/2^{128}$ with probability approximately

$$2\sum_{i\geq\delta}\binom{2^{128}}{2^{127}+i}/2^{2^{128}} \approx 1-\operatorname{erf}(\delta/\sqrt{2^{127}}) \approx \exp(-\delta^2/2^{127}),$$

where erf is the standard error function. For example, the attack demonstrates insecurity $\approx 2^{-65}$ with probability above 30%, and demonstrates insecurity $\approx 2^{-80}$ 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 any such collisions will tend to push δ away from 0, helping the attack.

3.2. Precomputing larger success probabilities. The same analysis applies to a modified attack D_s that appends a short string s to the AES outputs before hashing them: D_s demonstrates insecurity $\approx\delta/2^{128}$ with probability $\approx\exp(-\delta^2/2^{127})$. 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, $\delta = 2^{67}$: this attack D_s demonstrates insecurity $\approx 2^{-61}$ with probability $\approx 1-\operatorname{erf}(2^{3.5}) \approx 2^{-189}$. There are 2^{192} choices of 192-bit strings s, so presumably at least one of them will have D_s demonstrating insecurity $\approx 2^{-61}$. 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 $n \in \{0, 1, 2, \ldots, 64\}$, D_s demonstrates insecurity $\approx 2^{n-64}$ with probability $\approx 1 - \operatorname{erf}(2^{n+0.5}) \approx \exp(-2^{2n+1})$. 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 demonstrating insecurity $\approx 2^{n-64}$.

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 insecurity $\approx 2^{n-K/2}$, where K is the number of bits in the cipher key: query $2K$ bits of ciphertext, append s, and hash the result to 1 bit.

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 $\Theta(2^n)$ 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.

3.3. Iteration. High levels of insecurity 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 $f_7(k) = \operatorname{AES}_k(0) + 7$.

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 $\operatorname{AES}_k(1)$.

This computation finds 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 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 advantage 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 $2^K p$.

3.4. Multiple targets. Iteration becomes more efficient when there are multiple targets: U ciphertexts $\mathrm{AES}_{k_1}(0), \mathrm{AES}_{k_2}(0), \ldots, \mathrm{AES}_{k_U}(0)$ for U independent uniform random keys $k_1, k_2, \ldots, k_U$. Assume for simplicity that U is much smaller than 2^K; the hypothesis $U \leq 2^{K/4}$ 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 advantage 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 $2^{K/3}U^{-1/3}$; the attack then finds each key with probability $2^{-K/3}U^{1/3}$, with a cost of $2^{K/3}U^{-1/3}$ 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. Precomputation again has only a small benefit. There is still a benefit in success probability from handling U keys together: achieving success probability p costs essentially $(2^K/U)p$.

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

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

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

3.6. Previous work. All of the attacks described here have appeared before. In fact, when the conjectures in [7, Section 3.6] and [2, Section 3.2] were made, they were already inconsistent with known attacks.

The iteration idea was introduced by Hellman in [27] for the special case $U = 1$. Many subsequent papers 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 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 [29] recently considered the family of attacks D_s. They explained that there should be a short string s demonstrating insecurity $\approx 2^{-K/2}$, and analyzed the 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, [24, Section 7], which says that this is "well known in complexity theory".) De, Trevisan, and Tulsiani in [21] 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 [21, Section 1], but without any discussion of the resulting trouble for the literature on provable concrete secret-key security, and without any discussion of possible fixes.

Biham, Goren, and Ishai in [13, 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 $p > 2^{-K/4}U^{-1/2}$.

4 Breaking the NIST P-256 elliptic curve

This section analyzes the cost of an attack against NIST P-256 [35], 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.

4.1. 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 [37] of Pollard's rho method [38], 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 [42], 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.

4.2. 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 4.1. 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 constant probability, 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.

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 rather high 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 3.3 but much smaller than the precomputation in Section 3.2.

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 C points are precomputed, one should run approximately C walks in parallel with inputs depending on Q. The precom-

putation then covers $2^t C$ points, and the computations involving Q cover approximately $2^t C$ points, leading to a high probability of success when $2^t C$ reaches $\sqrt{\ell}$. The AT cost is also $2^t C$. This attack has the same cost as the standard Pollard rho method, except for small constants; there is no advantage in the precomputations.

4.3. Previous work. Kuhn and Struik in [30] 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 $c < 1$.

5 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 4, but this section assumes that the user has stopped this attack by increasing q to 384 bits (at a performance penalty).

5.1. 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, discussed in detail in Section 6, 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.

5.2. 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 $\le y$, 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 [28] and [19]. The gap between known algorithms and existing algorithms is thus even larger than indicated in this section. We emphasize that none of the algorithms perform well in the AT metric.

6 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.

6.1. NFS with precomputation. This attack is a variant of NFS, the standard attack against RSA. For simplicity this description omits several NFS optimizations. See [17] 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"

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

The standard estimate 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 $-H \le a \le H$, $0 < b \le H$, 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 discussed 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 [10]. 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) 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 [17] 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.

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

$$\begin{aligned} d &\in (1.1047\ldots + o(1))(\log N)^{1/3}(\log\log N)^{-1/3}, \\ \log m &\in (0.9051\ldots + o(1))(\log N)^{2/3}(\log\log N)^{1/3}, \\ \log y &\in (0.8192\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (0.8193\ldots + o(1))\log L, \\ \log H &\in (1.0034\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (1.0034\ldots + o(1))\log L. \end{aligned}$$

so that

$$\begin{aligned} u &\in (1.1047\ldots + o(1))(\log N)^{1/3}(\log\log N)^{-1/3}, \\ u\log u &\in (0.3682\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (0.3682\ldots + o(1))\log L, \\ d\log H &\in (1.1085\ldots + o(1))(\log N)^{2/3}(\log\log N)^{1/3}, \\ v &\in (2.4578\ldots + o(1))(\log N)^{1/3}(\log\log N)^{-1/3}, \\ v\log v &\in (0.8192\ldots + o(1))(\log N)^{1/3}(\log\log N)^{2/3} = (0.8192\ldots + o(1))\log L. \end{aligned}$$

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 $L^{0.8192\ldots+o(1)}$ 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 $L^{1.9760\ldots+o(1)}$.

6.3. 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 $\leq y$ (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 [11, 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 $B \approx 2^{67.69}$, and the total cost of smoothness detection for all $(a,b) \in F$ is approximately $2^{51.38}$ 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 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 $2^{121.41}$.

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 $2^{122.8}$.

6.4. Previous work. There are two frequently quoted cost exponents for NFS without precomputation. Buhler, Lenstra, and Pomerance in [17] obtained RAM cost $L^{1.9229\ldots+o(1)}$. Coppersmith in [20] introduced a "multiple number fields" tweak and obtained RAM cost $L^{1.9018\ldots+o(1)}$.

Coppersmith also introduced NFS with precomputation in [20], 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 [9] 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

[1] — (no editor), *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. Citations in this document: §1.1.

[2] Mihir Bellare, *New proofs for NMAC and HMAC: security without collision-resistance*, in Crypto 2006 [**25**] (2006), 602–619. URL: http://cseweb.ucsd.edu/~mihir/papers/hmac-new.html. Citations in this document: §1.1, §1.2, §1.3, §1.3, §3.6, §B.1.

[3] Mihir Bellare, Joe Kilian, Phillip Rogaway, *The security of cipher block chaining*, in Crypto 1994 [**23**] (1994), 341–358; see also newer version [**4**]. Citations in this document: §1.

[4] Mihir Bellare, Joe Kilian, Phillip Rogaway, *The security of the cipher block chaining message authentication code*, Journal of Computer and System Sciences **61** (2000), 362–399; see also older version [**3**]. ISSN 0022–0000. URL: http://www-cse.ucsd.edu/~mihir/papers/cbc.html. Citations in this document: §1.1, §1.1, §2, §2, §2.1, §2.1, §2.2, §B.2, §B.5, §B.6.

[5] Mihir Bellare, Phillip Rogaway, *Optimal asymmetric encryption — how to encrypt with RSA*, in Eurocrypt 1994 [**22**] (1995), 92–111. URL: http://cseweb.ucsd.edu/~mihir/papers/oaep.html. Citations in this document: §1.

[6] Mihir Bellare, Phillip Rogaway, *The exact security of digital signatures: how to sign with RSA and Rabin*, in Eurocrypt 1996 [**34**] (1996), 399–416. URL: http://www-cse.ucsd.edu/~mihir/papers/exactsigs.html. Citations in this document: §1, §1, §1, §1.1, §1.2, §1.3, §B.1.

[7] Mihir Bellare, Phillip Rogaway, *Introduction to modern cryptography*, 2005. URL: http://cseweb.ucsd.edu/~mihir/cse207/classnotes.html. Citations in this document: §1.1, §1.2, §1.3, §3.6, §B.1.

[8] Daniel J. Bernstein, *How to stretch random functions: the security of protected counter sums*, Journal of Cryptology **12** (1999), 185–192. URL: http://cr.yp.to/papers.html#stretch. Citations in this document: §B.6.

[9] Daniel J. Bernstein, *Circuits for integer factorization: a proposal* (2001). URL: http://cr.yp.to/papers.html#nfscircuit. Citations in this document: §6.4.

[10] Daniel J. Bernstein, *How to find smooth parts of integers* (2004). URL: http://cr.yp.to/papers.html#smoothparts. Citations in this document: §6.1.

[11] Daniel J. Bernstein, *Scaled remainder trees* (2004). URL: http://cr.yp.to/papers.html#scaledmod. Citations in this document: §6.3.

[12] Daniel J. Bernstein, *Proving tight security for Rabin–Williams signatures*, in Eurocrypt 2008 [**40**] (2008), 70–87. URL: http://cr.yp.to/papers.html#rwtight. Citations in this document: §B.6.

[13] Eli Biham, Yaron J. Goren, Yuval Ishai, *Basing weak public-key cryptography on strong one-way functions*, in TCC 2008 [**18**] (2008), 55–72. Citations in this document: §3.6.

[14] Peter van Emde Boas, *Machine models and simulation*, in [**32**] (1990), 1–66. Citations in this document: §2.1.

[15] Andrey Bogdanov, Dmitry Khovratovich, Christian Rechberger, *Biclique cryptanalysis of the full AES*, in Asiacrypt 2011 [**31**] (2011), 344–371. URL: http://eprint.iacr.org/2011/449. Citations in this document: §1.1.

[16] Richard P. Brent, H. T. Kung, *The area-time complexity of binary multiplication*, Journal of the ACM **28**, 521–534. URL: http://wwwmaths.anu.edu.au/~brent/pub/pub055.html. Citations in this document: §2, §2.3, §2.3.

[17] Joe P. Buhler, Hendrik W. Lenstra, Jr., Carl Pomerance, *Factoring integers with the number field sieve*, in [**33**] (1993), 50–94. Citations in this document: §6.1, §6.1, §6.4.

[18] Ran Canetti (editor), *Theory of cryptography, fifth theory of cryptography conference, TCC 2008, New York, USA, March 19–21, 2008*, Lecture Notes in Computer Science, 4948, Springer, 2008. ISBN 978-3-540-78523-1. See [13].

[19] An Commeine, Igor Semaev, *An algorithm to solve the discrete logarithm problem with the number field sieve*, in PKC 2006 [**44**] (2006), 174–190. Citations in this document: §5.2.

[20] Don Coppersmith, *Modifications to the number field sieve*, Journal of Cryptology **6** (1993), 169–180. Citations in this document: §6.4, §6.4.

[21] Anindya De, Luca Trevisan, Madhur Tulsiani, *Non-uniform attacks against one-way functions and PRGs*, Electronic Colloquium on Computational Complexity **113** (2009). Citations in this document: §3.6, §3.6.

[22] Alfredo De Santis (editor), *Advances in cryptology — EUROCRYPT '94, workshop on the theory and application of cryptographic techniques, Perugia, Italy, May 9–12, 1994, proceedings*, Lecture Notes in Computer Science, 950, Springer, 1995. ISBN 3-540-60176-7. MR 98h:94001. See [5].

[23] Yvo Desmedt (editor), *Advances in cryptology — CRYPTO '94, 14th annual international cryptology conference, Santa Barbara, California, USA, August 21–25, 1994, proceedings*, Lecture Notes in Computer Science, 839, Springer, 1994. ISBN 3-540-58333-5. See [3].

[24] Yevgeniy Dodis, John Steinberger, *Message authentication codes from unpredictable block ciphers*, in Crypto 2009 [**26**] (2009), 267–285. URL: http://cs.nyu.edu/~dodis/ps/tight-mac.pdf. Citations in this document: §3.6.

[25] Cynthia Dwork (editor), *Advances in cryptology — CRYPTO 2006, 26th annual international cryptology conference, Santa Barbara, California, USA, August 20–24, 2006, proceedings*, Lecture Notes in Computer Science, 4117, Springer, 2006. ISBN 3-540-37432-9. See [2].

[26] Shai Halevi (editor), *Advances in cryptology — CRYPTO 2009, 29th annual international cryptology conference, Santa Barbara, CA, USA, August 16–20, 2009, proceedings*, Lecture Notes in Computer Science, 5677, Springer, 2009. See [24].

[27] Martin E. Hellman, *A cryptanalytic time-memory tradeoff*, IEEE Transactions on Information Theory **26** (1980), 401–406. Citations in this document: §3.6.

[28] Antoine Joux, Reynald Lercier, *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. Citations in this document: §5.2.

[29] Neal Koblitz, Alfred Menezes, *Another look at HMAC* (2012). URL: http://eprint.iacr.org/2012/074. Citations in this document: §1.3, §1.3, §1.3, §3.6, §B.5.

[30] Fabian Kuhn, Rene Struik, *Random walks revisited: extensions of Pollard's rho algorithm for computing multiple discrete logarithms*, in SAC 2001 [**43**] (2001), 212–229. URL: http://www.distcomp.ethz.ch/publications.html. Citations in this document: §4.3.

[31] Dong Hoon Lee, Xiaoyun Wang (editors), *Advances in cryptology — ASIACRYPT 2011, 17th international conference on the theory and application of cryptology and information security, Seoul, South Korea, December 4–8, 2011, proceedings*, Lecture Notes in Computer Science, 7073, Springer, 2011. ISBN 978-3-642-25384-3. See [15].

[32] Jan van Leeuwen (editor), *Handbook of theoretical computer science, volume A: algorithms and complexity*, MIT Press, 1990. ISBN 0-262-22038-5. See [14].

[33] Arjen K. Lenstra, Hendrik W. Lenstra, Jr. (editors), *The development of the number field sieve*, Lecture Notes in Mathematics, 1554, Springer-Verlag, 1993. ISBN 3-540-57013-6. MR 96m:11116. See [17].

[34] Ueli M. Maurer (editor), *Advances in cryptology — EUROCRYPT '96: proceedings of the fifteenth international conference on the theory and application of cryptographic techniques held in Saragossa, May 12–16, 1996*, Lecture Notes in Computer Science, 1070, Springer, 1996. ISBN 3-540-61186-X. MR 97g:94002. See [6].

[35] National Instititue for Standards and Technology, *Digital signature standard*, Federal Information Processing Standards Publication 186-2 (2000). URL: http://csrc.nist.gov. Citations in this document: §4.

[36] Phong Q. Nguyen (editor), *Progress in cryptology — VIETCRYPT 2006, first international conference on cryptology in Vietnam, Hanoi, Vietnam, September 25–28, 2006, revised selected papers*, Lecture Notes in Computer Science, 4341, Springer, 2006. ISBN 3-540-68799-8. See [39].

[37] Paul C. van Oorschot, Michael Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), 1–28. ISSN 0933–2790. URL: http://members.rogers.com/paulv/papers/pubs.html. Citations in this document: §4.1.

[38] John M. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918–924. ISSN 0025–5718. MR 58:10684. URL: http://www.ams.org/journals/mcom/1978-32-143/S0025-5718-1978-0491431-9/S0025-5718-1978-0491431-9.pdf. Citations in this document: §4.1.

[39] Phillip Rogaway, *Formalizing human ignorance*, in VIETCRYPT 2006 [**36**] (2006), 211–228. URL: http://www.cs.ucdavis.edu/~rogaway/papers/. Citations in this document: §B.6.

[40] Nigel P. Smart (editor), *Advances in cryptology — EUROCRYPT 2008, 27th annual international conference on the theory and applications of cryptographic techniques, Istanbul, Turkey, April 13–17, 2008, proceedings*, Lecture Notes in Computer Science, 4965, Springer, 2008. ISBN 978-3-540-78966-6. See [12].

[41] Douglas R. Stinson, *Some observations on the theory of cryptographic hash functions* (2001). URL: http://eprint.iacr.org/2001/020. Citations in this document: §B.6.

[42] Edlyn Teske, *On random walks for Pollard's rho method*, Mathematics of Computation **70** (2001), 809–825. URL: http://www.ams.org/journals/mcom/2001-70-234/S0025-5718-00-01213-8/S0025-5718-00-01213-8.pdf. Citations in this document: §4.1.

[43] Serge Vaudenay, Amr M. Youssef (editors), *Selected areas in cryptography: 8th annual international workshop, SAC 2001, Toronto, Ontario, Canada, August 16–17, 2001, revised papers*, Lecture Notes in Computer Science, 2259, Springer, 2001. ISBN 3–540–43066–0. MR 2004k:94066. See [30].

[44] Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin (editors), *9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings*, Lecture Notes in Computer Science, 3958, Springer, 2006. ISBN 978–3–540–33851–2. See [19].

A Appendix: Graphs

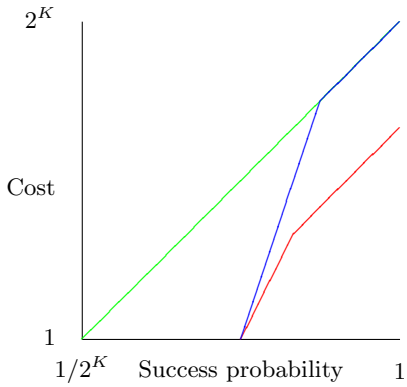

Fig. A.1. Cost summary of PRP attacks against one K-bit key. Horizontal axis: Attack success probability p, from $1/2^K$ to 1, on a logarithmic scale. Vertical axis: Attack cost, from 1 to 2^K, again on a logarithmic scale. Top curve (green): Cost $2^K p$, approximating cost of simple exhaustive search. Bottom curve (red): Cost $2^K p^2$ for $p \leq 2^{-K/3}$ and $2^{2K/3}p$ for larger p, approximating RAM/NAND cost of best attack known. Middle curve (blue, merging with green): Cost $2^{3K/2}p^3$ for $p \leq 2^{-K/4}$ and $2^K p$ for larger p, approximating AT cost of best attack known.

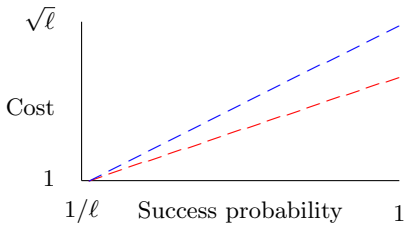

Fig. A.2. Cost summary of discrete-logarithm attacks for a group of size ℓ. Horizontal axis: Attack success probability p, from $1/\ell$ to 1, on a logarithmic scale. Vertical axis: Attack cost, from 1 to $\sqrt{\ell}$, again on a logarithmic scale. Bottom curve (red): Cost $(p\ell)^{1/3}$, approximating RAM/NAND cost of best attack known. Top curve (blue): Cost $(p\ell)^{1/2}$, approximating AT cost of best attack known.

B Appendix: Trying to salvage the insecurity metric

This appendix analyzes the merits of five possible responses to the gap between standard-definition insecurity and actual insecurity. None of the responses are completely satisfactory, but some of them are arguably better than others.

B.1. Response 1: circle the wagons. One possible response is to defend the metric, arguing that the attacks described in Sections 3, 4, 5, and 6 actually *should* be viewed as assigning security levels far below 2^{128} to AES, NIST P-256, DSA-3072, and RSA-3072. In other words, this response is that standard-definition insecurity *is* actual insecurity, and that taking precomputation into account would be understating actual insecurity.

This response has the virtue of minimizing the number of changes required to the literature. The other responses considered below require revisiting every proof to see whether the theorem can still be proven in a new metric; this response preserves the metric. Of course, the conjectures made in [6, Section 1.4], [7, Section 3.6], [2, Section 3.2], etc. would still have to be withdrawn.

The problem with this response is that it seems divorced from common sense. Why should cryptographers be more concerned about a time-2^{60} attack that takes time 2^{300} to find than about a time-2^{70} attack requiring no effort to find? Users aiming for the best possible security, subject to performance constraints, should prefer a system where the best attack is of the first type over a system where the best attack is of the second type; underestimating attack cost by ignoring precomputation will lead those users to select the wrong system, hurting their own security.

One might argue that precomputation should be ignored because it can be amortized across many targets. However, this argument confuses two different concepts. Non-uniform attacks and multiple-target attacks (and non-uniform multiple-target attacks) are often quantitatively and qualitatively different: non-uniform attacks often benefit from vastly larger precomputation (as illustrated by the doubly exponential cost $\exp(2^{2n+1})$ to find the attack D_s in Section 3.2), while multiple-target attacks often benefit from batching (as illustrated by Section 3.4).

B.2. Response 2: switch to the NAND metric. Another possible response is to change the algorithm cost model from the RAM metric to the NAND metric.

This response would cause trouble for the literature on provable concrete security. (All of the responses below would also cause various levels of trouble.) Proofs would have to be reviewed for RAM-dependent cost analyses, and many theorems would have to change, because many reductions would become much more expensive. For example, eliminating repeated queries to an oracle is a very common step in security proofs; it is practically free in the RAM metric (add each query into a fast associative array) but much slower in the NAND metric. In other words, even though the NAND metric was mentioned as an "alternative" in [4], the literature did not develop in a way consistent with this alternative.

The motivation for this response, as mentioned in Section 2, is the hope that this response would fix the "pathologies" in the RAM metric: i.e., that the gap between actual security and the standard definition of insecurity is an artifact of the low-cost random access provided by the RAM metric. However, the analyses in Sections 3, 4, 5, and 6 do not provide any support for this idea. All necessary random accesses appear in large batches, allowing reasonably efficient NAND computations.

B.3. Response 3: switch to the *AT* metric. Another possible response, analogous to the previous response but different in one critical detail, is to change algorithm cost model from the RAM metric to the AT metric.

Our cost analyses provide reason to hope that this response *does* fix essentially all of the pathologies in the RAM metric. There is an exception in one corner case — precomputation appears to help PRP attacks for probabilities below $2^{-K/4}U^{-1/2}$, where K is the key length

and U is the number of targets — but one can argue that such low-probability attacks are of no concern for cryptographic users.

This response causes trouble for the literature on provable concrete security, similar to the previous response but even more pervasive: even more theorems would have to change. Like the NAND metric, the AT metric makes serial random access much more expensive than the RAM metric; unlike the NAND metric, the AT metric makes a large batch of table accesses much more expensive than the RAM metric.

B.4. Response 4: add effectivity. In provable security (and in complexity theory more broadly) it is standard to formalize "one can find a cost-C algorithm A that breaks X" as "there exists a cost-C algorithm A that breaks X". This formalization ignores the question of how *difficult* it is to find A. Another possible response is to switch to another formalization that explicitly quantifies this difficulty.

The obvious way to quantify the findability of A is as the minimum cost required by all algorithms B that print A. The obvious objection is that there is always a cost-approximately-C algorithm to print A: namely, an algorithm that simply includes, and prints, a copy of A. However, one can easily exclude this trivial algorithm by allowing only *small* algorithms B.

Consider, for example, a large chip containing billions of standard AES key-search units. This chip breaks AES with AT cost roughly 2^{128}. The chip has a regular structure and area far below 2^{128}, so it is printed at moderate cost by an even smaller chip B. Similar comments apply to the standard chips to attack NIST P-256, DSA-3072, and RSA-3072 at cost roughly 2^{128}: all of them are printed at moderate cost by small chips B.

Consider, as another example, the hard-to-find attack A of Section 3.3, which finds an AES key with high probability using 2^{43} tables, each of size 2^{43}, for a total RAM cost on the scale of 2^{86}. The description of A in Section 3.3 is a small algorithm B that prints A, but B has RAM cost on the scale of 2^{128}. One can trade some space for time by embedding part of A into B, but as far as we know every algorithm B significantly smaller than A has negligible chance of printing A with RAM cost significantly below 2^{128}.

Consider, as a third example, the hard-to-find chip A of Section 3.2, with AT cost about 2^{3n}: area about 2^{2n} and time about 2^n. As far as we know, every significantly smaller chip B has negligible chance of printing A in any tolerable amount of time. This example suggests that it is possible to eliminate some corner pathologies that were not eliminated by merely switching to the AT metric.

Note that it is important to limit the cost of B in some reasonable cost metric, not merely the size of B. All of the precomputations considered in this paper can be carried out by rather small RAM algorithms; in other words, the outputs have low Kolmogorov complexity. For the same reason, the NAND metric is useless for measuring the cost of B.

In general, it seems reasonable to redefine the insecurity of X as the maximum, over all size-limited cost-limited algorithms B that print cost-limited algorithms A, of the probability that A succeeds in breaking X. (We emphasize that probability here considers not just the randomness in X and in A, but also the randomness in B; otherwise the best choice of B is a tiny algorithm that prints out random bits.) This definition is explicitly parametrized by three numerical limits, and implicitly parametrized by the metrics used to specify those limits.

This response stops all of the precomputations considered in this paper, although it still allows the attack considered in Section 3.1. Presumably this response will not exclude any attacks that humans have a reasonable chance to find: except for minor implementation

details, humans are simply chips, and rather small chips by cryptanalytic standards. Of course, one can imagine humans building larger chips that in turn find algorithms that the humans would not have found directly, and to model this one can consider longer chains such as algorithms that print larger algorithms that in turn print larger algorithms; but it seems reasonable to insist that the chain start with a small algorithm (small enough for humans to find) and to put a time limit on each algorithm.

This response, like the previous two responses, causes trouble for the literature on provable concrete security. Each theorem must be restated to track not only the cost of A but also the size and cost of B.

B.5. Response 5: add uniformity. The final response we consider is to eliminate non-uniform security definitions: to prevent precomputation by requiring a single attack algorithm to work against many different cryptographic systems.

The classic form of uniformity considered in the computational-complexity literature is size-uniformity. One considers, for example, an attack against the entire RSA family (a single algorithm that takes as input an RSA key of any length), not just RSA-3072. One defines the insecurity of RSA as the maximum success probability ϵ of any attack taking time at most t; here both ϵ and t are functions of the length of the RSA modulus.

Observe that this approach abandons the goal of defining, e.g., the insecurity of RSA-3072. Substituting 3072 into ϵ and t does not work: it allows exactly the same precomputations as in Section 6.

An alternative, already used in common definitions of collision resistance, is to consider uniformity across wider families of functions. There is no longer a definition of the security of AES; instead there is a definition of the security of a family of 2^{128} variants of AES. This security depends on the choice of family. One might try to define a family of functions "similar" to AES, hoping that the uniform security of this family reflects the actual security of AES; but cryptanalysts have little motivation to study the family, so building confidence is difficult. For RSA-3072 the situation is even worse: any reasonable family of 3072-bit functions arguably sharing the security of RSA-3072 seems to be vulnerable to the same precomputations as RSA-3072. For elliptic-curve cryptography the situation is somewhat better, since one can reasonably ask questions about the security of (e.g.) a random curve meeting the IEEE P1363 criteria over a randomly chosen 256-bit prime field; however, this is of no help in understanding the security of the NIST P-256 curve.

It is clear that insisting on enough uniformity, taking enough steps away from specific cryptographic primitives towards sufficiently diverse families of cryptographic primitives, would eliminate the gap analyzed in this paper. The gap relies on non-uniformity, and we have chosen to highlight non-uniformity in the title of this paper.

The fundamental problem with this response is that it disconnects provable security from cryptographic reality. For almost twenty years the literature on provable concrete security has promised to formally define and study the security of specific cryptographic systems of interest to cryptographic users and cryptanalysts, such as AES and AES-CBC-MAC and RSA-3072. Adding uniformity would abandon this promise. Without these definitions it is unclear how to make meaningful statements comparing the security of two different ciphers, or two different curves, or any two cryptographic protocols that are specific enough to actually be used in practice.

A second problem with this response is that it forces syntactic changes to most theorems. For example, instead of proving a theorem comparing the security of a block cipher F to the

security of CBC^m-F as in [4], one would have to prove a theorem comparing the security of a family of block ciphers to the security of the corresponding CBC family. This change requires abandoning some of the proofs in the literature, as pointed out by Koblitz and Menezes in [29], and might even require abandoning some theorems.

B.6. Recommendations. We believe that accurately modeling reality is more important than minimizing the number of changes required to the literature. We recommend switching to the AT metric (response 3), capturing real limitations on communication cost that are ignored by the RAM metric and the NAND metric. We also recommend adding effectivity (response 4), capturing the fact that attackers are limited in precomputation cost. We recommend against adding uniformity (response 5); users are in fact using AES and NIST P-256 and RSA-3072, not large families of variants of AES and NIST P-256 and RSA-3072.

We recommend stating provable-security theorems in a way that minimizes the hassle of switching to a new cost metric. For example, consider again the main theorem of [4], comparing the security of a block cipher F to the security of CBC^m-F. To prove this theorem one compares $R(A)$ to A in cost and in success probability, where A is any attack against CBC^m-F and R is a particular reduction producing an attack $R(A)$ against F. The comparison of success probability is independent of the cost metric, and we recommend stating it as a separate theorem that can be reused for different cost metrics. The following theorem from [12] illustrates how simple such statements can be:

"**Theorem 3.1.** $\mathrm{PrFactor}(\mathrm{RandSquare}(A)) \geq (1/2)\,\mathrm{PrInvBlind}(A)$."

The reduction RandSquare is defined before the theorem, and PrFactor and PrInvBlind are two types of success probabilities. This theorem is independent of cost metric, and is easy to reuse in various higher-level theorems that compare the insecurity of the objects attacked by A and $\mathrm{RandSquare}(A)$ in various cost metrics: the proof of a higher-level theorem analyzes the relative costs of A and $\mathrm{RandSquare}(A)$ and appeals to this theorem for the relative success probabilities.

This type of modularization of provable-security theorems can be traced back to at least 1999 (see [8, Theorem 4.1]), if not earlier, but at that point was not claimed to have any particular advantages. In 2001, Stinson proposed studying explicit hash-function reductions as a workaround for the separation between non-uniform collision resistance (which is negligible) and uniform collision resistance (which often appears to be very high); but Stinson's collision-resistance theorems (e.g., [41, Theorem 3.1]) do not actually make these reductions explicit and are trivialities as stated. Rogaway in [39] analyzed the same proposal much more carefully, and gave examples of nontrivial theorems about black-box and non-black-box reductions; but these theorems were still monolithic, handling probability together with a particular cost metric and encapsulating the reductions inside the proofs, so changing the cost metric is unnecessarily difficult.

Our analyses suggest that effectivity and the AT metric provide two levels of defense against the unrealistic attacks considered in this paper. However, we would not be surprised if further cost-model changes turn out to be desirable for other reasons, and we think that a modular style for provable-security theorems will reduce the effort required to make such changes in the future.