On Security Preserving Reductions – Revised Terminology

Oded Goldreich

Abstract. Many of the results in Modern Cryptography are actually transformations of a basic computational phenomenon (i.e., a basic primitive, tool or assumption) to a more complex phenomenon (i.e., a higher level primitive or application). The transformation is explicit and is always accompanied by an explicit reduction of the violation of the security of the complex phenomenon to the violation of the simpler one. A key aspect is the efficiency of the reduction. We discuss and slightly modify the hierarchy of reductions originally suggested by Leonid Levin.

Keywords: Foundations of Cryptography, Complexity, Reductions.

An early version of this article appeared as *ePrint* Report 2000/001. The current revision is quite minimal.

1 Introduction

Modern Cryptography is concerned with the construction of *efficient* schemes for which it is *infeasible* to violate the security feature. Thus, we need a notion of efficient computations as well as a notion of infeasible ones. The computations of the legitimate users of the scheme ought to be efficient, whereas violating the security features (via an adversary) ought to be infeasible. Our notions of efficient and infeasible computations are "asymptotic" (or rather functional):[1] They refer to the running time as a function of the security parameter. This is done in order to avoid cumbersome formulations that refer to the actual running-time on a specific model for specific values of the security parameter. Still, one can easily derive such specific statements from the asymptotic treatment.

Efficient computations are commonly modeled by computations that are polynomial-time in the security parameter. The polynomial bounding the running-time of the legitimate user's strategy is fixed and typically explicit and small (still in some cases it is indeed a valuable goal to make it even smaller). Here (i.e., when referring to the complexity of the legitimate user) we are in the same situation as in any algorithmic research. Things are different when referring to our assumptions regarding the computational resources of the adversary. A common approach is to postulate that the latter are polynomial-time too, where

[1] Actually, the term "asymptotic" is misleading, since from the functional treatment of the running-time (as a function of the security parameter), one can derive statements for *any* value of the security parameter.

the polynomial is *not* a-priori specified. In other words, the adversary is restricted to the class of efficient computations and anything beyond this is considered to be infeasible. Although many definitions explicitly refer to this convention, this convention is *inessential* to all known results (in the area). In all cases, a more general (and yet more cumbersome) statement can be made by referring to adversaries of running-time bounded by any function (or class of functions). For example, for any function $T:\mathbb{N}\to\mathbb{N}$ (e.g., $T(n) = 2^{\sqrt[3]{n}}$), we may consider adversaries that on security parameter n run for at most $T(n)$ steps. Doing so we (implicitly) define as infeasible any computation that (on security parameter n) requires more than $T(n)$ steps.

The results obtained in Modern Cryptography are in most cases conditional ones. That is, based on some relatively simple intractability assumptions (e.g., the existence of one-way functions [3, Chap. 2]) one constructs and establishes the security of more complex applications (e.g., unforgeable signature schemes [3, Chap. 6]). In many cases these results are stated in an oversimplified form, where a typical form reads *if the function f cannot be inverted in polynomial-time, then the scheme S_f (which utilizes f) cannot be broken in polynomial-time*. However, what is actually proved in such works is stronger. Typically, the proof of security of S_f specifies, for any function $T:\mathbb{N}\to\mathbb{N}$, a function $T':\mathbb{N}\to\mathbb{N}$ such that *if f cannot be inverted on n-bit images in time $T(n)$, then S_f cannot be broken on inputs of length m in time $T'(m)$*. Furthermore, typically, the relation between T' and T takes the form

$$T'(m) \;=\; \frac{p_2^{-1}(T(p_1^{-1}(m)))}{p_3(m)}\ , \tag{1}$$

where p_1, p_2, p_3 are some fixed polynomials. Such a relation results from the fact that the proof utilizes a reduction of the task of inverting f on strings of length n to the task of breaking S_f on strings of length $p_1(n)$. Thus, assuming on the contrary to the security claim that S_f can be broken in time $T'(m)$ on inputs of length $m = p_1(n)$, one obtains an algorithm inverting f on inputs of length n in time $T(n) \le p_3(p_1(n)) \cdot p_2(T'(p_1(n)))$.

It should be clear (and it is indeed well-known) that the aforementioned relation between T and T' determines the strength of the theoretical result as well as its potential practical applicability. Specifically, in almost all the cases the relation takes the form of Eq. (1), and in these cases one is interested in the specific polynomials p_1, p_2, p_3.

The purpose of this note is to discuss a popular classification of such reductions, attributed to Leonid Levin and presented in [12]. We suggest to modify this classification a little.

2 Preliminaries

Actually, the foregoing discussion is over-simplified, because it refers only to the running-time of the violating algorithms (and implicitly suggesting that we talk of algorithms that succeed always or almost always). In many cases, the statements are more complex, referring both to the running-time of algorithms and to a (probabilistic) measure of success. Two such common measures are

1. The success probability of easily verified events. For example, the success probability of an inverting algorithm (for a specific one-way function), or the success probability of a forging algorithm (for a signature scheme).
2. The gap in probability between two experiments. An archetypical example is the notion of computational indistinguishability. Here, for two distributions ensembles, $\{X_n\}$ and $\{Y_n\}$, we consider the gap between the probability that an algorithm A outputs 1 on input X_n and the probability A does so on input Y_n. Thus, definitions such as security of encryption schemes [7], pseudorandomness [1, 13, 4], and (computational) zero-knowledge [8] fall into this category.

The distinction between the foregoing two types is crucial for Levin's suggestion to incorporate the running-time and the success measure into a single measure (see Section 2.2). Note that in order to succeed with probability at least 2/3 in an attempt of the first type one has to repeat trying for $\Theta(1/\epsilon(n))$ times, where $\epsilon(n)$ is the success probability in a single attempt. On the other hand, in order to amplify a distinguishing gap of $\epsilon(n)$ into a gap of 2/3 we need to repeat the experiment(s) for $\Theta(1/\epsilon(n)^2)$ times.[2]

2.1 The general form of security reductions

Before presenting Levin's approach, let us present the general form that most results take. Typically, one starts with a basic primitive, denoted f (for sake of uniformity with the Introduction), and constructs a scheme S_f. (Each of the two is coupled with its own notion of violation, determining the measure of success.) The proof of security of S_f is by a reduction to violation of security of f. That is, such a proof shows, for any $t' : \mathbb{N} \to \mathbb{N}$ and $e' : \mathbb{N} \to \mathbb{R}$, how to convert an algorithm violating S_f with time complexity t' and success measure e' into an algorithm for violating f with time complexity t and success measure e. Calling the former an S_f-violator and the latter an f-violator, the conversion is by a reduction that typically specifies polynomials $p_1, p_2, ..., p_7$ such that on input of length n the f-violator invokes the S_f-violator on inputs of length $m = p_1(n)$, and satisfies $t(n) = p_2(t'(m)) \cdot p_3(1/e'(m)) \cdot p_4(m)$ as well as $e(n) = p_5(e'(m)) \cdot p_6(1/t(m)) \cdot p_7(1/m)$. It follows that, for any function $T:\mathbb{N}\to\mathbb{N}$ and $\epsilon : \mathbb{N} \to \mathbb{R}$, if f cannot be violated on n-bit inputs in time $T(n)$ with success measure $\epsilon(n)$, then S_f cannot be violated on m-bit inputs in time $T'(m)$ with success measure $\epsilon'(m)$, where T' and ϵ' may be any pair of functions satisfying

$$T(p_1^{-1}(m)) = p_2(T'(m)) \cdot p_3(1/\epsilon'(m)) \cdot p_4(m) \tag{2}$$

$$\epsilon(p_1^{-1}(m)) = \frac{p_5(\epsilon'(m))}{p_6(T(m)) \cdot p_7(m)} \tag{3}$$

[2] The above discussion refers to an abstract experiment (or pair of experiments). When applied to the examples given above, repeating the experiment means things like inverting a one-way function on one of several independently selected images, or distinguishing between multiple samples of two ensembles.

where $p_1, p_2, ..., p_7$ are the polynomials specified above. (Assuming, on the contrary, that S_f can be violated on m-bit inputs in time $T'(m)$ with success measure $\epsilon'(m)$, implies – via the reduction – violation of f on n-bit inputs in time $T(n)$ with success measure $\epsilon(n)$.)

2.2 Levin's notion of work

In order to simplify treatments as above, Levin suggested to incorporate the running-time and the success-measure of each violating algorithm into a single measure called work. The foregoing distinction between easily verifiable and non-verifiable success measures is crucial to his suggestion. For a verifiable success measure, the work of an algorithm A with running-time $t_A : \mathbb{N} \to \mathbb{N}$ and success measure $\epsilon_A : \mathbb{N} \to \mathbb{R}$ is defined as $w_A(n) \stackrel{\text{def}}{=} t_A(n)/\epsilon_A(n)$. For a (non-verifiable) success measure of the gap type, the work of an algorithm A with running-time $t_A : \mathbb{N} \to \mathbb{N}$ and success measure $\epsilon_A : \mathbb{N} \to \mathbb{R}$ is defined as $w_A(n) \stackrel{\text{def}}{=} t_A(n)/\epsilon_A^2(n)$. (We stress that the definition of work is problem specific and ad-hoc in nature.)[3]

In the sequel, we shall adopt Levin's simplification. A reader feeling uncomfortable with this choice, may consider only algorithms with constant success measure (in which case work is identical to time (up-to a constant factor)). Security will be defined as a (possibly postulated) lower bound on the work of violating algorithms. For example, one may assume that the security of factoring is $\exp(n^{1/4})$, and infer (based on this assumption)[4] that pseudorandom generators of security $\exp(n^{1/4})$ exist.

Definition (security): *Let Π be some primitive with an associated notion of violation that specifies a notion of success measure and induces a notion of work of violating algorithms. We say that Π has security $S : \mathbb{N} \to \mathbb{N}$ if any algorithm A violating Π has work function that grows faster than S.*

3 Levin's Hierarchy of Reductions (revisited)

In order to demonstrate the different quality of certain reductions, Levin has suggested three types of reductions, which were later canonized in Luby's book [12]. Letting $S : \mathbb{N} \to \mathbb{N}$ denote the security of the basic primitive, and $S' : \mathbb{N} \to \mathbb{N}$ the security of the complex primitive constructed from the former, the three types of reductions are:

(L1) A reduction is linearly preserving if it guarantees $S'(n) \geq S(n)/\text{poly}(n)$.

(L2) A reduction is polynomially-preserving if it guarantees $S'(n) \geq S(n)^e/\text{poly}(n)$, for some constant $e > 0$.

---

[3] The abstract discussion above does not fully justify the definition (see Footnote 2). Furthermore, other functionalities of running-time and success-measure may make sense too.

[4] See [2, Sec. 3.4].

(L3) A reduction is weakly-preserving if it guarantees $S'(n) \geq S(n^d)^e/\text{poly}(n)$, for some constants $d, e > 0$.

Levin has noted that, for nicely-behaved security measures, a reduction that guarantees $S'(n) \geq S(n/d)^e/\text{poly}(n)$, for some constants $d, e > 0$, is also polynomially-preserving. The argument is based on the fact that in our context all primitives are breakable within exponential time (i.e., time 2^n on input length n), and so one may assume without loss of generality that $S(n) \leq 2^n$. Furthermore, for "nicely-behaved" functions S, which are exponentially bounds, and for $c > 1$ one may expect that $S(cm) \leq S(m)^c$ holds. Thus, $S'(n) \geq S(n/d)^e/\text{poly}(n) \geq S(n)^{ed}/\text{poly}(n)$. Still, it seems inappropriate to identify the effect of e and d in a guarantee such as the foregoing (L2). Furthermore, when doing so, we lose an important distinction, which is represented in the gap between the following Types (T2) and (T3).

3.1 The revised hierarchy

(T1) A reduction is strongly preserving if it guarantees $S'(n) \geq S(n)/\text{poly}(n)$. (This is identical to (L1) above.)

(T2) A reduction is linearly-preserving if, for some constant $c \geq 1$, it guarantees

$$S'(n) \geq \frac{S(n/c)}{\text{poly}(n)}$$

(This extends (T1), where $c = 1$, in an important way.)

(T3) A reduction is polynomially-preserving if, for some constants $c \geq 1$ and $e > 0$, it guarantees

$$S'(n) \geq \frac{S(n/c)^e}{\text{poly}(n)}$$

(Formally, (T3) extends (L2), where $c = 1$; but, for "nicely behaved security measures" (see the foregoing discussion), type (T3) is equivalent to type (L2).)

(T4) A reduction is weakly-preserving if, for some constants $c, d, e > 0$, it guarantees

$$S'(n) \geq \frac{S(cn^d)^e}{\text{poly}(n)}$$

(This is equivalent to (L3) above.)

Thus, we replace (L2) by the two distinct categories (T2) and (T3).

3.2 Discussion

On the relation between (T2), (T3) and (L2). Levin's category (L2) is a special case of our (T3). In light of the discussion about, we believe that Levin himself would not care much about the extension of (L2) to (T3). In contrast, we believe that the distinction between Types (T2) and (T3) is very important.

We note that many claims made by Luby [12] regarding (L2) actually refer to either (T2) or (T3), and are valid for (L2) only under the above assumption (i.e., $S(cn) \leq S(n)^c$, for every constant $c > 1$) which collapses (T3) into (L2). Furthermore, when referring to (L2) one losses the important distinction between Types (T2) and (T3). These considerations are examplified by considering the following results.

- *A hard-core predicate for any one-way function* [6]: The original reduction of [6] (as well as the better known alternative reduction (as presented in [2, §2.5.2.1–3])) is of Type (T3).[5] In contrast, the improved reduction of Levin [11] (see also [2, §2.5.2.4]) is of Type (T2).
- *Security-preserving amplification of one-way function* [5]: The reduction demonstrating this result for the case of one-way permutations is of Type (T2). In contrast, the known reduction (of [5]) for the case of regular one-way functions is only of Type (T3), for some range of parameters.[6]

Thus, the distinctions between the strengths of the aforementioned pairs of results are reflected in the distinction between (T2) and (T3), but are not reflected by Levin's Hierarchy (since these results are all of type (L2)). We chose these examples because they are famous cases in which the entire point of the corresponding work is obtaining an improvement in quality of reductions among the studied primitives. Thus, the distinction between (T2) and (T3) is essential for making the point (as demonstrated above).

Beyond (T4). With the exception of a single case, all results we are aware of (in the field) are proven by a reduction of Type (T4), or lower. The only exception is Levin's observation regarding the existence of a *universal one-way function* (cf., [10] and [2, Sec. 2.4.1]).

A final warning. It should be clear that the above classification (as well as the one suggested in [12]) is ad-hoc in nature. Namely, it only represents our knowledge of the current reductions, and an attempt to classify them in a way that reflects their theoretical strength and practical applicability. Each type may be further refined according to the constants (and/or polynomials) appearing in its definition. Furthermore, in some cases (depending on such refinements), a reduction with higher type may be preferable (in practice) to one with lower type (e.g.,. $2^{\sqrt{n}} < n^{100}$ for $n < 10^6$).

[5] The claim in [12] by which the reduction is of type (L2) is correct only for "nicely behaved security measures" (see foregoing discussion).

[6] Actually, in the regular case, the construction in [5] depends on the security of the basic (weak) one-way function, and so we have a family of reductions one per each security function S (which needs to be efficiently computable). These reductions are of Type (T3), provided that, for some $d < 1$, $S(n) < 2^{n^d}$. Otherwise they are only of Type (T4).

An out of scope comment: As discussed in Footnote 6, some results are proven by a construction that depend on the security of the basic scheme; that is, for every security function S, a different construction of a complex primitive is presented (assuming that the basic one has security S). One should prefer results proven via a single construction, which is oblivious of the security of the basic scheme. The security of the resulting construct will depend on the security of the basic one, but the latter need not be known a-priori. In practical terms this means that one may make a weak assumption regarding the basic scheme such that this assumption guarantees sufficient security for the construct. If the basic scheme turns out to be more secure than originally assumed then the resulting construct will benefit in security (as per the security guarantee given with the reduction). In contrast, when the construction depends on the assumed security, better than postulated security of the basic scheme may not translate to better security of the construct.

Acknowledgments

We are grateful to Mihir Bellare for helpful comments.

References

1. M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. on Comput.*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.
2. O. Goldreich. *Foundation of Cryptography: Basic Tools.* Cambridge University Press, 2001.
3. O. Goldreich. *Foundation of Cryptography: Basic Applications.* Cambridge University Press, 2004.
4. O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *J. of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.
5. O. Goldreich, R. Impagliazzo, L.A. Levin, R. Venkatesan, and D. Zuckerman. Security Preserving Amplification of Hardness. In *31st FOCS*, pages 318–326, 1990.
6. O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st STOC*, pages 25–32, 1989.
7. S. Goldwasser and S. Micali. Probabilistic Encryption. *J. of Comp. and Sys. Sci.*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.
8. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. on Comput.*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985. Earlier versions date to 1982.
9. S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. on Comput.*, April 1988, pages 281–308.
10. L.A. Levin. One-Way Function and Pseudorandom Generators. *Combinatorica*, Vol. 7, pages 357–363, 1987.

11. L.A. Levin. Randomness and Non-determinism. *J. Symb. Logic*, Vol. 58(3), pages 1102–1103, 1993.
12. M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.
13. A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.