Non-Black-Box Techniques In Crytpography

Thesis for the Ph.D degree

Boaz Barak

Introduction

A *computer program* (or equivalently, an *algorithm*) is a list of symbols – a finite string. When we interpret a string Π as a program, we associate with this string a function that the string Π *computes*. For example, if we interpret the string

```
int f(int x) {
    return x+1;
}
```

as a program in the C programming language, then we associate with it the function $f(\cdot)$ where $f(x) \stackrel{def}{=} x+1$ for any integer x. Given a program Π and a value x, it is possible to compute $f(x)$, where $f(\cdot)$ is the function associated with the program Π, by executing the program Π on a computer. We sometimes call the string Π the *representation* or *code* of the function $f(\cdot)$.

Since a program is a string, it sometimes makes sense to use it as input to a different program. Indeed, algorithms that take other programs as input are very common in Computer Science. In most cases, these algorithms use their input program as a *subroutine*. By this we mean that, on input a program Π, the algorithm's operation does not depend on the particular representation of the program Π, but rather, the algorithm only uses its input program Π to evaluate the *function* that the program Π computes. That is, the algorithm only uses Π to obtain a "black box" for computing $f(\cdot)$, such that the algorithm can feed inputs to and receive outputs from this box. We call such an algorithm (that only uses black-box access to the program it gets as input) a *black-box* algorithm.

As mentioned above, black-box algorithms are very popular in computer science. Many times, when trying to solve a particular task, one would write an algorithm that solves the task if it is given black-box access to programs that solve some simpler tasks, and then write programs that solve these simpler tasks. This approach is a basic paradigm of software engineering, and almost all programming languages implement mechanisms (such as function calls) to facilitate it.

Black-box algorithms are also very popular in the more theoretical aspects of Computer Science. For example, when proving that a decision problem L is **NP**-complete, one needs to show the existence of a black-box algorithm B (where B is usually called a *reduction*), such that if B is given black-box access to an algorithm A that solves the problem L, then B can solve the SATISFIABILITY problem. Such reductions also appear in cryptography. For example, consider the constructions of public key encryption schemes whose security can be reduced to the factoring problem [Rab79, BG84]. This is shown by providing a black-box algorithm B, such that if B is given black-box access to an algorithm A that compromises the security of the encryption scheme, then B can solve the factoring problem.

Why are black-box algorithms so popular? The reason that black-box algorithms are so popular is that it seems very hard to make use of the particular representation of a program as a string. Understanding the properties of a function from the code of a program that computes it (also

known as *reverse-engineering*) is a notoriously hard problem. In fact, considerable efforts are made at writing programs so they would be easy to understand. Programs written without such efforts, or programs written or compiled to low level languages are considered to be quite incomprehensible. In fact, in some cases it is either *proven* (e.g., the HALTING problem, Rice's theorem) or widely conjectured (e.g., the SATISFIABILITY problem) that when trying to learn properties of a function, there is no significant advantage to getting its representation as a program, over getting black-box access to it. Thus, a common intuition is the following:

The only useful thing one can do with a program is to execute it (on chosen inputs).

In this thesis, we test this intuition in several settings in cryptography. Somewhat surprisingly, we find several cases in which it does *not* hold. That is, we find several cases in which it can be shown that *non-black-box* algorithms have significantly more power than *black-box* algorithms. We use this additional power of non-black box algorithms to obtain new results. Some of these results were previously proven to be *impossible* to obtain when using only black-box techniques.

1 Our Results

The main theme of this thesis is that non-black-box techniques *can* indeed be more powerful than black-box techniques in several interesting contexts in cryptography. Below, we elaborate more on our specific results in particular contexts. We only mention here our results, and we do not elaborate on the ideas and techniques used to obtain these results. Each chapter in this thesis has its own introduction, which contains a much more detailed (but still high level) discussion on the results of the chapter and on the ideas used in the proofs. We note that we present the results here in a different order than the order of the chapters. This is because the results of Chapter 6 are somewhat easier to state than the results of Chapters 4 and 5, but the proofs of Chapter 6 are actually more complicated and use some ideas from the previous chapters.

1.1 Code Obfuscation

Code obfuscation is about trying to make practical use of the difficulty of reverse-engineering programs. Informally, an obfuscator $\mathcal{O}$ is an (efficient, probabilistic) "compiler" that takes as input a program Π and produces a new program $\mathcal{O}(\Pi)$ that has the same functionality as Π yet is "unintelligible" in some sense. Obfuscators, if they exist, would have a wide variety of cryptographic and complexity-theoretic applications, ranging from software protection to homomorphic encryption to complexity-theoretic analogues of Rice's theorem. Most of these applications are based on an interpretation of the "unintelligibility" condition in obfuscation as meaning that $\mathcal{O}(\Pi)$ is a "virtual black box," in the sense that anything one can efficiently compute given $\mathcal{O}(\Pi)$, one could also efficiently compute given black-box access to Π.

Several constructions of software obfuscators have been previously suggested (c.f., [CTL97] and the references therein). However, no formal definition of obfuscation has been suggested, and so in particular none of these candidates has been proven to meet some formal security definition. In Chapter 3 of this thesis, we initiate a theoretical investigation of obfuscation. Our main result is that, even under very weak formalizations, obfuscation is *impossible*. We prove this by constructing a family of programs $\mathcal{P}$ that are *inherently unobfuscatable* in the sense that

1. There is an efficient algorithm A such that for every program $\Pi \in \mathcal{P}$ and every program Π' that computes the same function as Π, $A(\Pi') = \Pi$. That is, for every possible obfuscator $\mathcal{O}$, A can recover the original source code of Π from $\mathcal{O}(\Pi)$.

but

2. When given only *black box* access to a program Π chosen at random from $\mathcal{P}$, it is infeasible to compute the program Π.

We extend our impossibility result in a number of ways, including even obfuscators that **(a)** are not necessarily computable in polynomial time, **(b)** only approximately preserve the functionality, and **(c)** only need to work for very restricted models of computation ($\mathbf{TC_0}$). We also rule out several potential applications of obfuscators, by constructing "unobfuscatable" signature schemes, encryption schemes, and pseudorandom function families.

1.2 Non-black-box Proofs of Security

A typical cryptographic theorem has the following form "Scheme X (e.g., a secure voting protocol) is as secure as Problem Y (e.g., factoring random Blum integers)". This statement means that if there exists an efficient algorithm A that breaks the security of the scheme X, then there exists an efficient algorithm B that can solve the problem Y. In all previous cases that we are aware of, such statements were proven via *black-box reductions*. That is, to show that the statement is true, one gave a construction of a generic algorithm B that takes as input both an instance of the problem Y and uses *black-box* access to an algorithm A. Then, one proves that if A is an algorithm to break the scheme X, then when given access to A, Algorithm B solves the problem Y.

A natural question is whether one can gain more power by using a *non-black-box* reduction. That is, whether by considering also reductions that let the algorithm B use also the *code* of A one can obtain new cryptographic schemes. In Chapter 6 we give a positive answer to this question. We use there a *non-black-box* proof of security to construct the first constant round non-malleable commitment scheme and the first constant-round non-malleable zero-knowledge argument system, as defined by Dolev, Dwork and Naor [DDN91].

Non-malleability is a strengthened notion of security that is needed for some applications of secure protocols. In particular, non-malleable commitment schemes capture better the intuitive notion of "digital envelopes" in the sense that the committed value is not only hidden from the receiver of such a commitment scheme but also the receiver cannot form a commitment to any related value. In contrast, when using a standard (i.e., malleable) commitment scheme, it may be the case that the receiver of a commitment to a value x, can form a commitment to a value related to x (e.g., $x+1$) even though he cannot learn x.

Previous constructions of non-malleable commitment schemes and zero-knowledge proofs either used a non-constant number of rounds, or were only secure under stronger setup assumptions (such as the availability of a public string that is chosen at random and published by a trusted third party).

As an intermediate step we define and construct a constant-round non-malleable *coin tossing* protocol. This coin-tossing protocol may be of independent interest.

1.3 Non-Black-Box Simulation

The *simulation paradigm*, introduced by Goldwasser, Micali, and Rackoff [GMR85], is one of the most important paradigms in the definition and design of cryptographic primitives. For example, this paradigm arises in a setting in which two parties, Alice and Bob, interact in some secure protocol (e.g., a zero-knowledge proof) and Bob knows a secret. We want to make sure that Alice hasn't learned anything about Bob's secret as the result of this interaction, and do so by showing that Alice could have *simulated the entire interaction by herself*. Therefore, she has gained no

further knowledge as the result of interacting with Bob, beyond what she could have discovered by herself.

Formally, this is shown by exhibiting a *simulator*. A simulator is an algorithm, that gets as input an algorithm A^* which describes Alice's strategy in the protocol, and outputs a distribution that is indistinguishable from the distribution of the messages that Alice sees in a real interaction with Bob when she is using the strategy A^*. The existence of such a simulator demonstrates that regardless of the strategy A^* that Alice's uses, she has not learned anything about Bob's secret that she couldn't have learned by herself without having any interaction with Bob (by simply running the simulator).

Almost all previously known simulators used only *black-box* access to the algorithm A^* they received as input.[1] In Chapter 4, we present the first construction of a protocol with a *non-black-box* simulator under standard assumptions. Using these new non-black-box techniques we obtain several results that were previously shown to be impossible to obtain using black-box simulators.

Specifically, assuming the existence of collision-resistent hash functions, we construct a new zero-knowledge argument (i.e., a computationally-sound proof) for any language in **NP** that satisfies the following properties:

1. It is zero-knowledge with respect to non-uniform adversaries with auxiliary information.
2. It has a constant number of rounds and negligible soundness error.
3. It remains zero-knowledge even if executed concurrently n times, where n is the security parameter. We call a protocol that satisfies this property a *bounded concurrent zero-knowledge* protocol.[2]
4. It is an Arthur-Merlin (public coins) protocol.
5. It has a simulator that runs in *strict* probabilistic polynomial-time, rather than *expected* probabilistic polynomial-time.

The above protocol should be contrasted with the following impossibility results regarding *black-box* zero-knowledge arguments for non-trivial languages: Goldreich and Krawczyk [GK90] showed that such protocols cannot satisfy both Properties 2 and 4. Canetti, Kilian, Petrank and Rosen [CKPR01] showed that such protocols cannot satisfy both Properties 2 and 3. In Chapter 5, we also show that such protocols cannot satisfy Properties 2 and 5.

In addition, in Chapter 5 we use this zero-knowledge system to obtain other new results in cryptography. These applications include **(a)** a construction of constant-round zero-knowledge argument of knowledge with a strict polynomial-time knowledge extractor, **(b)** a zero-knowledge resettably sound argument for NP, and **(c)** a resettable zero-knowledge argument of *knowledge*. We show that all these three applications are impossible to obtain when restricted to black-box techniques.

We remark that application **(b)** in particular, is somewhat counter-intuitive, and demonstrates well the power of non-black-box techniques. In particular it means that if you are given a device (e.g., a smart-card) that proves some statement σ in this system then you are not able to learn

[1] One exception is the zero-knowledge proof system of [HT99]. However, that protocol was constructed under a computational assumption that the authors themselves describe as unreasonable.

[2] The choice of n repetitions is quite arbitrary and could be replaced by any *fixed* polynomial (e.g. n^3) in the security parameter. This is in contrast to a standard concurrent zero-knowledge protocol [DNS98, RK99] that remains zero-knowledge when executed concurrently any polynomial number of times.

anything new about σ except its validity by "playing" with the device – feeding it with different inputs, and examining its outputs. However you can still be certain that if you were to open the device and examine its internal workings, you would be able to extract a witness for σ.

Relation to non-black-box proofs of security. We remark that in some sense non-black-box simulation is a special case of non-black-box proofs of security. This is because one can view the existence of a simulator as a proof that the protocol is secure, and so a non-black-box simulator can be viewed as a non-black-box proof of security. Indeed, usually when a zero-knowledge protocol is used as a component in a larger sub-protocol, the security reduction for the larger protocol involves using the simulator for the zero-knowledge protocol. Thus, if the zero-knowledge protocol used has a non-black-box simulator, then the security reduction for the larger protocols will be non-black-box.

References

[BG84] M. Blum and S. Goldwasser. An *Efficient* Probabilistic Public-Key Encryption Scheme which Hides All Partial Information. In *Crypto '84*, pages 289–299, 1984. LNCS No. 196.

[CKPR01] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. Record 2001/051, Cryptology ePrint Archive, June 2001. Extended abstract appeared in STOC' 01.

[CTL97] C. Collberg, C. Thomborson, and D. Low. A Taxonomy of Obfuscating Transformations. Technical Report 148, University of Auckland, July 1997. See also http://www.cs.arizona.edu/~collberg/Research/Obfuscation/index.html.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437 (electronic), 2000. Preliminary version in STOC 1991.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero Knowledge. In *Proc.* 30*th STOC*, pages 409–418. ACM, 1998.

[GK90] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Comput.*, 25(1):169–192, Feb. 1996. Preliminary version appeared in ICALP' 90.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. Preliminary version in STOC' 85.

[HT99] S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. Cryptology ePrint Archive, Report 1999/009, 1999. http://eprint.iacr.org/.

[Rab79] M. O. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, Jan. 1979.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt '99*, 1999. LNCS No. 1592.