A "proof-reading" of some issues in cryptography

Ivan Damgård

Department of Computer Science, BRICS, University of Aarhus

Abstract. In this paper, we identify some issues in the interplay between practice and theory in cryptography, issues that have repeatedly appeared in different incarnations over the years. These issues are related to fundamental concepts in the field, e.g., to what extent we can prove that a system is secure and what theoretic results on security mean for practical applications. We argue that several such issues are often overlooked or misunderstood, and that it may be very productive if both theoreticians and practitioners think more consciously about these issues and act accordingly.

1 Introduction

Design of systems using cryptography is a delicate, error-prone and difficult task. Despite the fact that cryptography is probably the part of general IT security that we understand best, people frequently get it wrong. There are many examples of this, for instance the design of the encryption for the GSM mobile phone system, where parity bits for error correction were added before the encryption, making cryptanalysis much easier [4]. Or a previous version of the PKCS#1 standard for RSA encryption where it was overlooked that error messages produced by the decryption algorithm could reveal information that an adversary could exploit [2].

There are several reasons for this state of affairs. Of course, simple lack of expertise is one reason, but more fundamentally, what appears to be common sense and reasonable to the designer may only be reasonable with respect to the attacks that the designer can imagine, whereas of course an attacker in real life may well do things that are completely unexpected. Second, a system may be reasonable in the application scenario that the designer has in mind, whereas systems in reality are often applied in contexts that the designer never anticipated.

As a concrete illustration, let us assume we design a protocol for exchanging confidential data based on RSA encryption. The hope is that to break the protocol, one has to invert the RSA encryption function, i.e., given public key n, e and ciphertext $x^e \bmod n$, find x. But as experience shows, there is a risk that we have overlooked shortcuts that the adversary can exploit to break the system easily, without having to invert the RSA function.

A well-known and very useful way to tackle these problems is to provide a rigorous proof that if an attacker could break the cryptosystem or protocol in question, then he could efficiently solve a computational problem that we can reasonably assume is hard (the RSA problem in our example). In this way we have assurance that no shortcuts have been overlooked in our system. This approach, which originates in the work of Goldwasser and Micali [9], is in our opinion one of the most productive and valuable contributions theoretical cryptography has to offer. Nevertheless, the actual substance of this contribution is surprisingly often misunderstood, over- or underestimated, to the extent that the author of this paper has felt it has become necessary to clarify a few issues in this context.

2 What does "Provable Security" mean?

The way to argue about security we have sketched above is often called "provable security", but this terminology is somewhat misleading, for (at least) four reasons:

First, a proof of the form just sketched does not actually prove security. It only proves that an efficient algorithm for successfully attacking the system would imply an efficient algorithm for solving the underlying computational problem. Granted, this contradicts the assumption that the underlying problem is hard - but unfortunately, we do not know how to prove that any concrete problem really is hard to solve in the sense that would be required here. In the following, we will therefore use the term *security reduction*, since the essential part of the proofs we consider is a reduction that takes an algorithm for attacking a system and builds from it an algorithm for solving a computational problem.

A second reason is that one may get may get the incorrect impression that once we have "proved security", we are home free. In reality, the situation is more complex: what a security reduction proves more precisely is that if we can break the system in time T, then we can solve the underlying problem in time $f(T)$, for some function f. Ideally, we want the reduction to be tight, that is, $f(T)$ is approximately equal to T. This allows us to draw the strongest possible conclusion: if our assumption on the underlying problem says that it cannot be solved in time less than some bound B, then $f(T)$ and hence T must larger than B. In our RSA example, if we choose a large enough modulus so that the best known algorithms for breaking RSA would take time corresponding to, say 2^{80} elementary operations on a computer, then with a tight reduction we can say that an adversary will also take time 2^{80} to break our system or he will have to find a completely new factoring algorithm. Some security reductions are not that good, we may only know that, e.g., $f(T) = T^2$. This means that we need to have larger key sizes before the reduction allows us to conclude that T is sufficiently large A non-tight security reduction does not, however, mean that a larger key size is *necessary.* Maybe a tight reduction exists that we just haven't found yet.

A third issue is the exact nature of the underlying problem: if the assumption that the problem is hard is to be credible, the problem needs to be as simple, natural and well studied as possible. If we are not careful when making assumptions, we risk making useless statements that are not much better that saying "the system is secure under the assumption that it is secure".

A final issue is that security is always proved in a specific model, where we make assumptions on what the adversary can do and what information is available to him. For instance, we may try to prove that our RSA based protocol is secure under chosen message attacks, where the adversary can submit any input he likes to the decryption algorithm and see what output is produced. The goal is then to decrypt some target ciphertext that was not decrypted"for free". This model assumes, for instance, that the adversary has no information on the internal state of the decryption algorithm. In some cases, such information may be (partially) available, for instance if the adversary can measure radiation from the decryption equipment. This can be modeled too, using the notion of physically observable cryptography [11], for instance. But one has to be aware that some model must be chosen in order to give a proof, and a protocol may well be secure in one model and not in another.

To summarize, we have pointed out that:

- A proof of security never proves security in an absolute sense, it relates security to an unproven assumption that some computational problem is hard.
- The quality of a security reduction should not be ignored – it matters how tight it is, and how strong the underlying assumption is.
- A security reduction only proves something in a particular model specifying what the adversary has access to and can do.

These observations are not new at all. They have been made many times, e.g., by Bellare [1], and also in a recent paper by Koblitz and Menezes (K&M)[10]. The interesting question is, however, what conclusions we should draw? how should cryptographers and practitioners act, having understood these facts? Some researchers, including K&M, take a very critical stand on "provable security". Provoked, perhaps, by the limitations we pointed out above, they find that the whole approach is of questionable value, and in particular researchers who let their work be guided by the goal of showing security reductions for their systems are, in fact, misguided. The argument seems to be that people build unnecessarily complicated and contrived systems only to be able to provide a security reduction for them, and also that people wrongly accept even very inefficent security reductions as an argument for believing in the security of a system.

In our opinion, such a critique misses several important points. A first high-level comment is that, while the security reduction technique indeed does not allow us to draw simple conclusions such as "it is secure", this does not mean that security reductions are not useful. What it does mean is that reality is more complex than we would perhaps have liked it to be, but this is not something we can blame on our scientific methods. In the following subsections, we comment in more detail on why we believe security reductions are useful despite their limitations.

2.1 Even inefficient security reductions are useful.

Consider again our running example, an RSA-based protocol for exchanging confidential data. Such a protocol, namely an older version of the PKCS#1 standard, was broken under a chosen ciphertext attack by Bleichenbacher [2], as mentioned above. However, his attack did not break RSA at all, in fact the whole point of the attack was to sidestep the RSA problem and instead exploit a design failure in the protocol. Hence, Bleichenbachers attack is of no help at all towards inverting the RSA function. Now, if there had been a security reduction for PKCS#1, relating its security on the RSA function, this would have meant that ANY polynomial time attack would also be a polynomial time algorithm for inverting RSA, even if the reduction had not been tight. Hence any such reduction would have excluded attacks such as Bleichenbachers. The conclusion is that if a system has a non-tight security reduction to a well studied and standard problem, this does not help us towards choosing key sizes for the system, but it does imply that the design has no short-cuts allowing attacks that side-step the problem we were trying to base the system on. This is a very useful piece of information, also in practice: from experience, design errors usually materialize exactly as attacks that circumvent the hard problem we were trying to use.

2.2 Security Reductions should be a design goal.

It has become standard in most cryptographic research to give security reductions for any construction that is proposed. Consequently, researchers are usually not satisfied with a construction where they cannot make a reduction go through, and will try to modify their ideas until the proof works, and often this concern influences already the early phases of a research project.

As mentioned, some researchers object to this way of working because they find that it leads to unnatural and unnecessarily complicated systems. So is it be better to first build a "natural" system and then try to prove something about it? A first comment is that being "natural" is very subjective notion and hardly one that can be used a guideline for designing cryptographic constructions. Futhermore, even if one accepts that being "natural" is a possible design goal, there is no compelling reason why this would exclude the existence of a security reduction.

Granted, sometimes there seems to be a cost involved in having a security reduction – some extra ingredient that the designers add, apparently only to make the reduction go through. But on the other hand, this might just as well be exactly the ingredient that prevents a devastating attack – an attack that even the designers were not aware of! As long as no one has proved that such an attack does not exist, we simply dont know whether the extra ingredient is superfluous or essential to security.

We believe that the only reasonable approach is to construct cryptographic systems with the objective of being able to give security reductions for them. Of course, we should not be happy about any reduction, we should strive for reductions that are efficient, and reduce to as weak assumptions as possible. And of course, we want as simple and efficient systems as possible. But on the other hand, we should not settle for protocols just because we think they "look natural" and "seem to be secure". We return to this issue in more detail after we have looked at an example in the next section.

3 An example: the adaptive adversary

To further illustrate the issues discussed above, we explain an example problem in more detail, and draw some conclusions for practice and theory that hopefully apply also beyond the example itself.

Consider an adversary who monitors the traffic on a network. This traffic is encrypted using public-key cryptography, we assume that each machine on the net has a private/public key pair set up, so every message is encrypted under the public key of the receiver. At the end of the day, based on what the adversary has seen, he decides to break into one of the computers on the net[1]. We will assume that this gives him access to the private information on that machine, including the private key, so he can now decrypt all messages intended for this machine. Based on what he now knows, he may decide on another machine to break into, etc. His resources do not, however, allow him to break into all machines. An attack as we have sketched here is called *adaptive* because the adversary decides where to break in adaptively during his attack, instead of making all decisions initially.

We will assume that the public-key cryptosystem used is secure (we discuss below what exactly this should mean) and that keys for different machines have been independently generated. If A is a subset of the machines, M_A will denote the set of all message sent to machines in A. The question now is: what can the adversary learn from such an attack? Clearly, from observing the traffic itself, he will be able to identify the sender and receiver of each message, and learn its length. It is also clear that if the adversary has broken into a subset A of machines, he will learn M_A. But can we conclude that he can learn ONLY M_A? or in other words, can the adversary get only the information that is obviously available?

It is very tempting to conclude that the answer is clearly yes: since keys are independent, seeing private keys of machines in A tells you nothing about the other private keys, and any reasonable notion of security for the cryptosystem should mean that if you do not know the private key, seeing the ciphertext tells you nothing about the plaintext.

A moments reflection will show, however, that the question may not be quite so straightforward. If the above intuition is good, then this should mean that there is some well-defined notion of security, such that if the cryptosystem satisfies it, then the adversary can learn no extra information. But existing definitions

[1] We assume for simplicity that first, all messages are sent and then the adversary starts breaking into machines. One may of course consider more general models, but this makes no essential difference to our discussion.

of security of public-key encryption talk about only a single public/private key pair, where the private key is (of course) never revealed to the adversary. In our case, we have a set of many keys, where a priori *every* private key may potentially be revealed. Although only a subset is revealed in every concrete instance of the game, even the adversary may not know in advance which subset it will be (since this depends on information he will see later). Maybe this is of no real consequence, but it does demonstrate that we need a proof to be sure about our conclusion – at least if we want to base ourselves on a simple assumption that we can understand and have confidence in.

One way to approach the problem is to note that ideally, we want that the adversary learns only the plaintexts he can trivially decrypt. In other words, it should be as if he has access to an oracle O telling him those plaintexts. More precisely, we define O such that on input the name of a machine n in the network, it will return $M_{\{n\}}$. If we can show that in a real attack, the adversary learns nothing more than he could learn from sending the names of corrupted machines to O, this will certainly be sufficient. This can be done by building a simulator S, which on one side talks to O and on the other side to the adversary. Whenever the adversary wants to break into a machine, we allow S to send the name of this machine to O and get its messages. Towards the adversary, S must simulate everything the adversary would see in a real attack, including all the ciphertexts, in such a way the adversary cannot distinguish this from a real attack. If this is the case, we have certainly demonstrated that the adversary learns no extra information: everything he sees, in particular the ciphertexts, can be convincingly simulated based only on the information he was supposed to learn.

Unfortunately, an attempt to build such a simulator immediately runs into a problem: S must initially simulate the ciphertexts that A expects to see. At this point, S does not know any plaintexts: it is not allowed to access O before the adversary breaks into a machine. It therefore seems it can do nothing but create some ciphertext on its own. Let one of these ciphertexts be $c_i = E_{pk_i}(m)$, where we assume it is intended for machine number i, with public key pk_i. S has created c_i as an encryption of message m. If the adversary later breaks into machine i, S will learn from O a message m_0, the message that was really sent. But now it is too late! When the adversary decrypts c_i, he will get m as result and not m_0. This simulation strategy therefore does not work: what the adversary sees is clearly different from the real view. No simple way to fix the problem is known. For instance, it is completely unreasonable to hope that S can guess all plaintexts correctly ahead of time.

There is a solution, however, which was found by observing that the problem comes from the fact that with standard public key encryption, S effectively commits to a decryption result by giving a ciphertext to the adversary. The idea is therefore to build a so-called *non-committing* encryption scheme. Here, we design the encryption process such that S can create a fake ciphertext c_i that looks like a real one, but does not contain any particular plaintext. When later S is given the right message m_0, it can create secret information related to c_i, so it looks as if m_0 was in fact the encrypted plaintext. This exactly solves the problem and allows the simulation to go through.

The known implementations of non-committing encryption require interaction between sender and receiver and generation of new key material for every message sent. They are therefore much less efficient than standard encryption schemes. It can even be shown that the interaction is unavoidable: no non-interactive, non-committing encryption scheme exists [12].

So are the extra complications involved in using non-committing encryption really necessary? There are two possibilities:

- The initial intuition about the system is correct, and the adversary really cannot learn extra information, even if standard public-key cryptography is used. This may be the case - the fact that the first attempt at a simulation did not work only shows that this particular proof technique fails, not that the result is false. Maybe there is a different simulation strategy, or a proof that is not based on simulation at all?
- The initial intuition is wrong, and there is in fact an attack which can only be prevented if we use non-committing encryption. This too is entirely possible: knowing the history of the field, one would have to be very naive to believe that because we cannot think of an attack, no attack exists.

What should theoreticians do in such a situation? Skeptics of provable security may tend to think that non-committing encryption is just an unnecessary complication and that standard encryption is of course good enough. On the other hand, some theoreticians may conclude that "the book is now closed" because non-committing encryption solves the problem. In our opinion, both attitudes are wrong: there is still an open problem left! Namely, show that non-committing encryption is necessary or that standard encryption suffices. This is all the more fascinating as a basic research problem because the intuition we have not been able to prove seems so compelling.

What about the practical side? Many practitioners would probably tend to trust the intuition and harvest the efficiency advantage in using standard public-key encryption. Indeed, this is what is happening in practice. It is very important to emphasize that there is nothing "wrong" or "illegal" about this, even from a theoretical point of view. *Not as long as you understand the extra risk that is being taken*, namely that the second possibility above turns out to be the truth. After all, any practical application takes risks, for instance the risk that the underlying computational assumption is false. However hard it may be, one always has to estimate the risk taken and balance it against the practical advantages offered by the design. And it is important that theoreticians help as much as possible to identify and estimate the risks, even if it means making guesses beyond what we know for sure.

3.1 Do random oracles help?

There is a somewhat surprising connection between the example just discussed and the random oracle model. This also deserves a discussion here, since the random oracle model has been the subject of much debate and is very central in the interplay between theory and practice.

In this model one assumes that all players have oracle access to a random function R. This means that any player can submit an input x to the oracle and will get $R(x)$ back. Moreover, the only way to get an output is to explicitly access the oracle and no one has any information about the value $R(x)$ until after someone has sent x to the oracle.

This model was introduced by Bellare and Rogaway [3] in order to formalize the intuition behind various applications of hash functions. When we use a hash function H, for instance as proposed by Fiat and Shamir [7] to build a signature scheme from an interactive protocol, there seems to be no way that an adversary could exploit the internal structure of H to attack the system. More concretely, if H is sufficiently complex, it seems that the attacker would have to treat H as if it was a random oracle. If this really is the case, we may as well assume that the system is actually run in the random oracle model, i.e., we may replace H by R.

In the random oracle model, one can prove that the Fiat-Shamir construction and many other constructions are secure. Unfortunately, this does not imply that these systems are secure in the real world. At most, it guarantees security from a certain class of attacks that treat the hash function as a black-box. But you do not get security in general because a concrete hash function is of course not a random oracle: as soon as you have specified the function, all outputs are fixed and predictable.

It is known that a security reduction in the random oracle model is not always useful. Several results have demonstrated that there exist systems which are secure in the random oracle model, but become insecure as soon as we replace R by a concrete function, no matter which function we use – see, e.g., [6, 5]. On the other hand, these example have been specifically engineered to demonstrate the difficulty of instantiating the random oracle by a specific function: the system tries to detect whether it is in the random oracle model or in the real world (by interacting with R or H) and "on purpose" does something that is insecure if it finds it is in the real world.

These results therefore do not show that the Fiat-Shamir construction is insecure in practice, for instance[2]. Indeed, one might still claim that if we use our hash function in a "sensible" way that is not designed to break down in the real world, everything should be fine, and we should be happy about a security proof in the random oracle model.

However, an observation by Jesper Buus Nielsen [12] shows that there is good reason to be skeptical about this point of view. He shows an extremely simple way to build a non-committing encryption scheme based on a random oracle: Suppose we have a public/secret key pair (pk, sk) for a cryptosystem that is secure in the standard sense. Then, to encrypt message m, we choose a random

[2] There is work showing that the Fiat-Shamir construction is insecure in a very particular context [8], but this is again a "specially engineered scenario". The practical proposals for using Fiat-Shamir remain unbroken.

value r and the ciphertext is the pair $(E_{pk}(r), R(r) \oplus m)$. To decrypt using sk, one decrypts the first part to get r, calls the oracle to get $R(r)$ and xor's with the last part to get m.

In the random oracle model, this is a non-committing encryption scheme in the sense described above: in the simulation proof from the previous section, whenever the simulator S needs to show a ciphertext to the adversary, it can do the following: choose values r, s at random and let the "ciphertext" be $(E_{pk}(r), s)$. Note that S has no particular message in mind at this point, but nevertheless what is produced looks exactly like a valid ciphertext. When later S wants to claim that some particular message m_0 was encrypted, it uses the fact that because r was random and has been encrypted under a secure encryption scheme, the adversary does not know r. And so with overwhelming probability, no one has yet called the oracle with input r. Therefore, $R(r)$ is undefined at this point, so S is free to define that $R(r) = m_0 \oplus s$. This is indeed a random value, and exactly makes it seem as if m_0 was the encrypted message.

But note that this scheme is non-interactive! And as mentioned above, no non-interactive, non-committing encryption scheme exists *in the real world*, where random random oracles are not available. This immediately implies that no matter which concrete function we use in place of R, the resulting scheme does not work anymore. Therefore, this construction is only secure in the random oracle model, there is no way we can use it in a real application.

What does this tell us? One can hardly claim that the scheme we have just seen was designed to make the random oracle model look bad. In fact it looks quite natural, at least at first sight. The lesson to learn is that deciding whether a scheme in the random oracle model has a chance of being secure in real life is not a trivial problem. It also emphasizes that, while having a security reduction in the random oracle model is better than nothing and can be useful as a sanity check, having one in the real world is highly preferable. It is therefore a well justified research goal to build constructions that work without random oracles, but are as efficient as those that need them.

4 Conclusion

Many cryptographic constructions that are used in practice have a theoretical justification that leaves something to be desired. Perhaps there is no security reduction, or the one we have is not tight, or it only works in the random oracle model. Any such application takes a risk, beyond the obvious one, that the basic cryptography may not be secure. Theoreticians should not dismiss such applications as "bad practice", but should instead make an effort to try to help estimate or reduce the risk that is being taken. This can be done by improving the constructions and/or the security reductions, but also by coming up with completely new solutions. Even if a theoretically nice but impractical solutions exists, this is not a good reason to stop working on the problem.

On the other hand, it is equally important that practitioners begin to understand that the theory of cryptography cannot always deliver simple answers such as “it is secure” or ‘it isn’t”. And that hence using cryptography as a black box, with no knowledge at all about what is inside, may be convenient but also potentially harmful to the security of your system. And finally, that our knowledge about the security of even well known schemes is not static, but develops over time, and this may require changes to practice that may not be convenient, but nevertheless necessary to decrease the security risks you are taking.

References

1. Mihir Bellare: Practice-oriented provable-security. Proceedings of First International Workshop on Information Security (ISW 97): 221-231. Springer Verlag LNCS.
2. Daniel Bleichenbacher: Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. Proceedings of CRYPTO 1998: 1-12. Springer Verlag LNCS.
3. Mihir Bellare, Phillip Rogaway: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. ACM Conference on Computer and Communications Security 1993: 62-73.
4. Elad Barkan, Eli Biham, Nathan Keller: Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. Proceedings of CRYPTO 2003: 600-616. Springer Verlag LNCS.
5. Mihir Bellare, Alexandra Boldyreva, Adriana Palacio: An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. EUROCRYPT 2004: 171-188. Springer Verlag LNCS.
6. Ran Canetti, Oded Goldreich, Shai Halevi: The Random Oracle Model Revisited, Proceedings of STOC 1998.
7. Amos Fiat, Adi Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. Proceedings of CRYPTO 1986: 186-194. Springer Verlag LNCS.
8. Shafi Goldwasser and Yael T. Kalai: On the (In)security of the Fiat-Shamir Paradigm, Proc. of FOCS 2003.
9. Shafi Goldwasser, Silvio Micali: Probabilistic Encryption. J. Comput. Syst. Sci. 28(2): 270-299 (1984).
10. Neal Koblitz and Alfred Menezes: Another look at “Provable Security”, J.Cryptology, 20(1), 2007, pp. 3-37. Springer Verlag.
11. Silvio Micali, Leonid Reyzin: Physically Observable Cryptography (Extended Abstract). Proceedings of TCC 2004: 278-296. Springer Verlag LNCS.
12. Jesper Buus Nielsen: Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. Proceedings of CRYPTO 2002: 111-126, Springer Verlag LNCS.