RZ 3076 (#93122) 23/11/1998
Computer Science/Mathematics 24 pages

Research Report

Why Chosen Ciphertext Security Matters

Victor Shoup

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

IBM
Research Division
Almaden · Austin · Beijing · Haifa · T.J. Watson · Tokyo · Zurich

Why Chosen Ciphertext Security Matters

Victor Shoup

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

This article motivates the importance of public-key cryptosystems that are secure against chosen ciphertext attack, and of rigorous security proofs. It also discusses the new cryptosystem developed by Cramer and Shoup, and its relevance in this regard.

1 Introduction

Security engineers often think of encryptions as a kind of "secure envelope," which only the proper addressee can open. This is a very compelling metaphor, and it is the inspiration for the design of numerous cryptographic protocols.

Like philosophers, those who work in the area of mathematical cryptography ask the question: so what do we *really* mean by a "secure envelope," and do such things *really* exist?

As one might expect, the engineers usually have no time or patience for the slow, and seemingly esoteric ponderings of the mathematicians. Sometimes, this is of course justified, but at other times, this attitude can lead to trouble. It was perhaps just such an attitude that led to the design flaws in the widely-used internet security protocol SSL that left it open to the attack discovered earlier this year by Daniel Bleichenbacher [6].

There have been, of course, many other attacks on networked computers, but typically, these attacks go after what are usually "weak links" that have nothing to do with cryptography, e.g., bugs in the operating system or communications software. But Bleichenbacher's attack is a direct attack on what is supposed to be the "strongest link": the security protocol and the underlying encryption function, a particular variant of RSA. In fact, Bleichenbacher's attack demonstrates that these encryptions *do not behave at all* like "secure envelopes."

In this article, we review some basic concepts from the theory of cryptography, relating to the security of cryptosystems as used in SSL. We will motivate this theory, and show why it is actually very important for real-world security concerns.

In particular, we will try to make some sense out of the notion of a "secure envelope." This notion is really an ideal that we cannot hope to implement completely. However, as we will see, we can isolate the *essential characteristic* of a secure envelope that can (in principle, at least) be implemented and which is necessary and sufficient for the design and analysis of higher-level protocols that use encryptions as "secure envelopes." This essential characteristic is called *security against chosen ciphertext attack* (or equivalently, *non-malleability*). As we shall see, even achieving this is not at all easy to do.

In this regard, we will also discuss the new cryptosystem of Cramer and

Shoup [10]; namely, it it the first *practical* cryptosystem that offers a mathematical *guarantee* of security against chosen ciphertext attack.

Our main goal here is to convince the reader that security against chosen ciphertext attack is important, as is a mathematical proof of such security. Sometimes these issues are not taken seriously, apparently for two main reasons:

- chosen ciphertext attacks that break a system in any meaningful way are too hard for an attacker to mount;

- protecting against chosen ciphertext attack, especially in a mathematically provable sense, is too expensive.

In short, the cure is worse than the disease. We hope to demonstrate that this is not so—that the "disease" is indeed quite serious, and the the "cure" is not so bad.

Organization

The rest of this paper is organized as follows. In §2, we recall the basic concepts of public key cryptography, and discuss the analogy of encryptions with idealized "secure envelopes." In §3, we discuss the plain-old RSA cryptosystem, and illustrate why it fails to approximate the ideal of a "secure envelope." In §4 we discuss several "fixes" to plain-old RSA, and their shortcomings, including a high-level description of Bleichenbacher's attack. In §5, we discuss formal, mathematical definitions of secure encryption that approximate the ideal of a "secure envelope." In §6, we discuss methods for rigorously analyzing the security of cryptosystems, and explain the meaning of the phrase "provably secure," as used by mathematical cryptographers. In preparation to the discussion of the Cramer-Shoup cryptosystem in §8, we discuss the simpler ElGamal cryptosystem in §7. In §9 we end with some brief concluding remarks.

2 Public Key Cryptography

We focus here only on *public key cryptography.* A person or computer "Bob" has a *public key*, which is known to the world, and a *private key*, which is known only to Bob. These two keys are usually generated by some random process, or algorithm, and although the two keys are correlated in some way, it should be difficult to compute the private key from the public key.

Now, a person/computer "Alice" wants to send a message m to Bob over the Internet, say. She *encrypts* the message, scrambling it in a certain way, producing a *ciphertext* ϵ. The computation of ϵ uses Bob's public key, which everyone knows, including Alice. Alice then sends ϵ over the Internet to Bob. After receiving ϵ, Bob can *decrypt*, or unscramble ϵ, recovering Alice's message m. Bob does this using his secret key. Since only Bob knows this secret key, only Bob can unscramble the message (or so the theory goes).

Note that it is the *receiver*, Bob, who has a key: the *sender*, Alice, does not require a key.

As mentioned in the introduction, these ciphertexts, or encryptions, are often thought of as "secure envelopes": Alice "locks" a message in an envelope, sends the envelope to Bob, and Bob opens it, reading the message. Because the envelope is "secure," only Bob can open the envelope with his key.

This is a very compelling and convenient metaphor, and security engineers often use this metaphor as a design principle when designing high-level protocols.[1]

However, an encryption scheme can at best only approximate a secure envelope. This is fundamentally because ciphertexts are *bit strings* (electronically represented) and not physical envelopes made out of bits of paper. This is of course obvious and trivial, but there are several quite serious problems that can potentially arise because of this.

- First, the bit string representing a ciphertext can be observed by an eavesdropper. An ideal "secure envelope" leaks *no information* about the message it contains. For example, if Alice sends two messages to Bob using "secure envelopes," then an eavesdropper can not tell if these two messages are equal or not. The same should hold, then, for an encryption scheme. This requirement already rules out any encryption scheme that is *deterministic*, i. e., always encrypts the same message the same way.

[1] Arguably, a better metaphor might be a "locked box," since real-world envelopes are not too difficult to open. Nevertheless, we follow tradition, and use the phrase "secure envelope," imagining that these are very strong and difficult to open.

- Second, ciphertexts can easily be replicated, whereas a physical envelope cannot be easily replicated. There is really nothing we can do about this, and higher level protocols that use encryptions must deal with the fact that this can happen.

- Third, ciphertexts can be easily modified, creating other ciphertexts. Since a ciphertext is just a bit string, we can do many things to it, like flip some bits from '1' to '0', or *vice versa*. For example, there are encryption schemes that leak no information about a message, but flipping a bit of the ciphertext effectively flips a bit of the message. Such a scheme is called "malleable," and cannot be tolerated in many applications. This undesirable property has no counterpart in the world of ideal "secure envelopes."

- Fourth, *any* bit string is potentially the encryption of *some* message. This fact can be exploited by an attacker that mounts an "active attack" in which the attacker does more than just eavesdrop, but actively participates in a protocol, sending its own messages to other parties. For example, an attacker could send arbitrary bit strings to Bob, and Bob would try to decrypt them, thinking that these bit strings are ciphertexts. Bob may react in a number of different ways when it tries to decrypt such a "ciphertext," and to the extent that the attacker can observe these reactions, it can gain information that could conceivably help it "crack" the cryptosystem. Such an attack is called a *chosen ciphertext attack*, and also has no counterpart in the ideal world of "secure envelopes."

  This type of attack may at first seem somewhat esoteric, but it was precisely with such an attack that Bleichenbacher broke SSL.

It turns out that (when properly formulated) security against chosen ciphertext attack and non-malleability are equivalent. It also turns out that this is the *essential characteristic* of secure envelopes that is necessary and sufficient in the design and analysis of protocols that treat encryptions like "secure envelopes."

We will illustrate these difficulties with "plain-old RSA," which suffers from all of the above-mentioned difficulties: it leaks partial information, it is malleable, and it is vulnerable to chosen ciphertext attack.

3 Plain-old RSA

In this section, we illustrate the difficulties of building a secure envelope by showing how a particular cryptosystem, while apparently offering some degree of privacy, does not act in any way like a secure envelope.

The cryptosystem we examine is "plain-old RSA"; that is, RSA as it was more or less originally proposed by Rivest, Shamir, and Adleman [21].

In the RSA scheme, the public key consists of a large integer n and an integer e. The integer n is the product of two large, randomly chosen primes, p and q. Typically, p and q have about 150 decimal digits or so, and so n has around 300 decimal digits. It turns out that generating these random primes and multiplying them together is relatively easy to do (for a computer, that is). But if we should throw p and q away, and keep only n, then there is no practical way to find p and q again. That is, there is no practical computer program that takes n as input, and factors it, computing p and q.

The secret key for the RSA scheme is an integer d such that $e \cdot d - 1$ is a multiple of $(p-1) \cdot (q-1)$. It turns out that such a d can be computed efficiently knowing p and q, and conversely, this d cannot be computed efficiently without factoring n.

The RSA encryption algorithm works with numbers in the set

$$\mathbf{Z}_n = \{0, \ldots, n-1\},$$

and performs multiplication on these numbers "mod n." This means that to multiply two numbers a and b in $\mathbf{Z}_n$, we form their ordinary product $c = a \cdot b$, and then "reduce" c mod n, which means we subtract a suitable multiple of n from c so that we end up with a number r that is back in $\mathbf{Z}_n$. This number r is called the product of a and b mod n, and can be computed quite efficiently. Besides modular multiplication, we can also efficiently perform modular addition, subtraction, and division.

We can also efficiently exponentiate mod n. If we want to compute x^e mod n, we could obviously do this with about e multiplications mod n. But in fact, this can be done much faster, with the number of multiplications being proportional to the number of *digits* in e—a much smaller number.

The point of using modular arithmetic is twofold: first, numbers don't get too big, because they are always reduced, and second, it is difficult to "undo" certain modular operations. For example, it turns out that although it is easy to square a number mod n, it is hard to take a square root mod n, without knowing p and q.

As mentioned above, for any x in $\mathbf{Z}_n$, we can efficiently compute x^e in $\mathbf{Z}_n$. The mystery of d is now explained: it turns out that for every x in $\mathbf{Z}_n$, $(x^e)^d = x$; that is, exponentiation by d "undoes" exponentiation by e.

Now, Bob's public key consists n and e, and his secret key is d. If Alice has a message m to send to Bob, we assume that m can viewed as an element of $\mathbf{Z}_n$; after all, m is just some string of bits or characters that can also be interpreted as a number. Then the encryption of m is

$$\epsilon = m^e \bmod n.$$

Upon receiving ϵ, Bob decrypts it by computing

$$\epsilon^d \bmod n = (m^e)^d \bmod n = m.$$

We should make one further remark on the values of e and d. The value e is fairly arbitrary. In fact, it is sometimes advocated to choose $e = 3$ for efficiency reasons. Regardless of the value of e, the value d in general is about the same size as n.

Given our current knowledge of algorithms, it seems a safe bet that if an eavesdropper sees a ciphertext ϵ corresponding to a *random* message m, then it will be effectively impossible for that eavesdropper to figure out what m is. Note that there may be other ways to recover m from its encryption besides factoring n; however, at the moment the only way known how to recover m is to factor n. It is a widely held and reasonable belief that recovering m from its encryption is difficult—this belief is known as the *RSA assumption*.

However, as we will now illustrate, even if the RSA assumption is true, plain-old RSA cannot possibly qualify as a "secure envelope." In particular, we will illustrate that plain-old RSA

- does not hide partial information,
- is "malleable,"
- is insecure against chosen ciphertext attack.

As these examples will illustrate, just the assumption that factoring is hard, or even the assumption that decrypting random encryptions is hard, is not enough to justify the use of plain-old RSA as a "secure envelope."

3.1 Plain-old RSA does not hide partial information

Suppose Alice wants to send orders to her stock broker Bob. An eavesdropper would like to know Alice's order. Furthermore, suppose the eavesdropper has good reason to believe that m is one of the three messages:

- m_1 = "buy IBM"
- m_2 = "sell IBM"
- m_3 = "hold IBM"

The eavesdropper can compute the encryptions $\epsilon_1, \epsilon_2, \epsilon_3$ of these three messages for himself, and when Alice sends an encryption of one of these three messages, say it is ϵ_2, the eavesdropper will know that Alice's message is m_2.

This example illustrates that plain-old RSA leaks partial information about messages, and and a "secure envelope" should not allow this.

3.2 Plain-old RSA is malleable

Suppose Alice wants to submit a number m, representing a bid, to Bob. Bob is accepting many bids, and will choose the lowest bid.

Suppose Alice's competitor want to underbid Alice by 10%. Here is how he can do this. We need to make the reasonable assumption that Alice bids in round amounts—multiples of 10. Alice's competitor intercepts Alice's encrypted bid ϵ, and computes

$$\epsilon' = \epsilon \cdot (9/10)^e \bmod n.$$

The decryption of ϵ' is the number

$$m' = 0.9m.$$

In this way, Alice's competitor can underbid Alice by 10%, without knowing anything about the actual value of Alice's bid!

This type of weakness is an example of "malleability," and is a weakness that a "secure envelope" should not have.

3.3 Plain-old RSA is insecure against chosen ciphertext attack

Plain-old RSA is also insecure against chosen ciphertext attack, which is a vulnerability that a "secure envelope" should not have.

In this type of an attack, the attacker wants to decrypt a "target" ciphertext ϵ. However, Bob—for some presumably good reason—will not give this decryption to the attacker. However, the attacker may be able to trick Bob into decrypting *other* ciphertexts. In particular, the attacker can generate a ciphertext

$$\epsilon' = \epsilon \cdot x^e \bmod n$$

for some number x, and if he can get Bob to decrypt this for him, obtaining m', then Bob can compute

$$m = m'/x \bmod n.$$

4 Can plain-old RSA be fixed?

We've identified several weaknesses in plain-old RSA that show that it does not even come close to the ideal of a "secure envelope." At this point, the reader may very well be thinking: haven't we just set up a *straw man*? Indeed, plain-old RSA is never used in practice, precisely because of these well-known weaknesses. Instead, what people actually use is plain-old RSA with a few modifications that attempt to fix these problems. We examine here some of the popular "fixes," and indicate their shortcomings.

4.1 Random Encoding

One idea that is often advocated to improve the security of plain-old RSA is to use a randomized "encoding" or "padding" scheme. That is, we encrypt m as

$$\epsilon = f(m, r)^e \bmod n,$$

where $f(m, r)$ encodes the message m using some random bits r. We stress that f is *not* a cryptographic encoding: it is easy for *anyone* to compute m from $f(m, r)$.

The hope is that this enhancement improves the security of RSA. However, as we shall see, if one is not extremely careful, one may actually *decrease* the security of RSA.

One simple way to define $f(m, r)$ is just to concatenate the two bit strings m and r. This is a popular idea. RSA, Inc. has a very popular encryption function, called PKCS #1, which—up until very recently—did essentially this. This encryption function is used by SSL, the security protocol that is widely used on the Internet. Bleichenbacher's attack on SSL, mentioned in the Introduction, is actually a chosen-ciphertext attack on RSA's PKCS #1.

At a high level, this attack works as follows. Suppose the attacker wants to decrypt ϵ. Then the attacker sends many ciphertexts of the form

$$\epsilon' = \epsilon \cdot x^e \bmod n,$$

for randomly chosen numbers x in $\mathbf{Z}_n$ to Bob, who is now a server, say, on the Internet. Upon receiving ϵ', Bob will compute, as usual,

$$a = (\epsilon')^d \bmod n.$$

Now Bob tries to apply the decoding function to a; that is, Bob tries to find m' such that $f(m', r) = a$ for some r. It will sometimes be the case that a is a proper encoding of a message, and sometimes not. All the attacker needs is this "error code" for a large, but reasonable, number of ciphertexts. Given just these error codes, Bleichenbacher shows how a very clever program can then recover the decryption of ϵ itself.

How feasible is Bleichenbacher's attack? In his paper, Bleichenbacher reports on experiments that suggest that between 300,000 and two million chosen ciphertexts must be sent to a server to decrypt a single message. That may seem like a lot, but it is still quite feasible. Although there are a number of minor changes that can be made to the SSL protocol to foil Bleichenbacher's attack, that is not the point: one would have to already suspect a weakness in the encryption scheme before bothering to implement such second-level defenses.

Bleichenbacher's attack is not the only known attack on such random encoding schemes. Along different lines, Coppersmith [9] showed in 1996 that a slightly different, but natural, randomized encoding scheme would leave RSA (with $e = 3$) open to the following attack: given two different encryptions of the same message, the attacker can compute the message.

The attacker here is completely passive—he only needs to eavesdrop and obtain the two different encryptions of the same message.

The above attacks show that random encoding is no panacea.

In 1994, Bellare and Rogaway [5] introduced a new encoding scheme for RSA, called OAEP (Optimal Asymmetric Encryption Padding), that uses a cryptographic hash function (like MD5 or SHA-1). This seems like a very robust encoding scheme, and there are no known attacks against it. Moreover, Bellare and Rogaway give a heuristic argument that this scheme is secure in a formal sense (see §5). While this argument is somewhat compelling, it is not fool-proof: there could *still* be unforeseen attacks on OAEP that are more efficient than factoring n.

The heuristic argument given by Bellare and Rogaway relies on "magical" properties of the hash functions: these properties are so magical that in fact no hash function could actually have these properties. Although their "magic hash function" argument is not entirely satisfying, it is still a big step in the right direction. In response to Bleichenbacher's attack, RSA's PKCS #1 is quickly being converted over to OAEP, although this could have—and probably should have—been done long ago.

4.2 Hybrid Schemes

Another approach to strengthening the security of RSA is to use a so-called "hybrid scheme." In such a scheme, the basic RSA function is used to encrypt a key that is then used in a conventional symmetric key cryptosystem, like DES, to encrypt the actual message. This approach can also be combined with the randomized encoding approach.

Hybrid schemes were initially introduced to improve efficiency. However, it was also noticed that they seem to have the added benefit of making it harder to mount certain types of attacks. Or at least, it is less obvious how to do so. It is entirely possible that there are unforeseen attacks that are more efficient than factoring n.

5 What is a secure cryptosystem?

We've seen several ways in which plain-old RSA fails to be a secure envelope. We've also seen that several attempts to "fix" plain-old RSA do not work.

And we've seen several "fixes" that seem to work better—at least, they have not yet been broken.

Given this somewhat unsettling state of affairs, it would be nice to have a cryptosystem that could be *guaranteed* to be secure, at least assuming the underlying mathematical problem (i.e., factoring large numbers) is indeed hard. This can indeed be done, but to do so, we need to have a good *definition* of security.

A significant amount of work has gone into just finding the "right" definition of security for a public-key cryptosystem. What makes a definition the "right" one? Well, there is no definitive answer to this question, but there are a number of important criteria by which a definition can be judged:

- it seems to capture our intuition;
- it rules out all known forms of attacks;
- it is *robust*, i.e., it can be formulated in a number of natural and equivalent ways;
- it is *useful*, i.e., the definition is adequate to be applied in proving the security of many high-level protocols that use encryption as a "primitive."

Of all these criteria, perhaps the last one is the most important.

In the presence of a *passive*, or *eavesdropping* adversary, the "right" definition is known as *semantic security*. Intuitively, this just means that an eavesdropper can get no partial information about an encryption. In the extreme case, even if an adversary knows that Alice is sending an encryption of one of just two possible messages ("buy IBM" or "sell IBM"), then an analysis of the ciphertext will not yield any information about which of the two was actually sent.

We can reasonably paraphrase the formal definition of semantic security as follows. It is defined in terms of a game which a "bad guy"—or adversary—plays against the "good guy."

- First, the good guy generates a public key in the prescribed manner, and gives this to the bad guy.
- Second, the bad guy computes two messages m_0, m_1 and gives these to the good guy.

- Third, the good guy flips a coin: if it is "heads," he encrypts of m_0; otherwise, he encrypts m_1. The good guy then gives this encryption to the bad guy. Note that the good guy flips the coin in secret, and does not directly reveal the outcome of the coin flip.

- Fourth, upon seeing this encryption, the bad guy outputs his guess as to the outcome of coin toss above.

Of course, by just guessing, the bad guy is correct with probability 1/2. Semantic security means: no *efficient* bad guy can guess correctly with probability *significantly* greater than 1/2. Here, *efficient* and *significantly* are actually technical terms that require further definition, but we won't get into that here.

The precise mathematical definition of this concept was first formulated and published by Goldwasser and Micali [16] in 1984, and is broadly accepted as the "right" definition of security in the presence of a passive adversary.

Nevertheless, it was known that in order to approximate the ideal of a "secure envelope," one has to deal with stronger, *active* adversaries: not just adversaries who can listen to messages sent by others over the network, but that can play an active role, sending their own messages to others. Bleichenbacher's attack on RSA is an example of such an "active attack."

The "right" formal, mathematical definition of security against active attacks evolved in a sequence of papers by Naor and Yung [19], Rackoff and Simon [20], and Dolev, Dwork and Naor [13]. The notion is called *chosen ciphertext security* (or equivalently, *non-malleability*).

The intuitive thrust of this definition is that even if an adversary can get arbitrary ciphertexts of his choice decrypted, he still gets no partial information about *other* encrypted messages.

As with semantic security, we can paraphrase the formal definition of security against chosen ciphertext attack as a game between a bad guy and the good guy.

- First, the good guy generates a public key in the prescribed manner, and gives this to the bad guy.

- Second, the bad guy asks the good guy to decrypt a number of ciphertexts; he is allowed to submit any ciphertext ϵ and see its decryption.

- Third, the bad guy computes two messages m_0, m_1 and gives these to the good guy.
- Fourth, the good guy flips a coin: if it is "heads," he encrypts of m_0; otherwise, he encrypts m_1. The good guy then gives this encryption, call it ϵ', to the bad guy.
- Fifth, the bad guy is again allowed to submit a number of arbitrary ciphertexts ϵ to the good guy for decryption, as above, subject only to the (obviously necessary) restriction that $\epsilon \neq \epsilon'$.
- Finally, the bad guy outputs his guess as to the outcome of coin toss above.

Chosen ciphertext security means: no *efficient* bad guy can guess correctly with probability *significantly* greater than $1/2$.

This is a *very* strong notion of security; so strong in fact, it might seem like overkill. However, it is in fact generally agreed to be the "right" definition. It seems that chosen-ciphertext security is the best approximation we have to ideal "secure envelopes." Also, it is very useful in the design and analysis of protocols: one can usually design a high-level protocol using "secure envelopes," and then instantiate these envelopes with a chosen-ciphertext secure cryptosystem, and the result is a secure protocol. See, for example, the protocols in [3] for key exchange, the protocols in [1] for escrow and fair exchange of digital signatures. It seems likely that in the future, as more people understand the concept of chosen ciphertext security, more high-level protocols will be designed and rigorously analyzed assuming this property about the underlying cryptosystem.

6 How do we know that a cryptosystem is secure?

So now we have formal mathematical definitions of security. And we have seen that many variations of RSA are actual *insecure*, even though the problems of factoring n or computing eth roots mod n remain difficult.

So how can we design a cryptosystem and *know* that it is indeed secure?

6.1 The *ad hoc* approach

Throw in some random padding here, some hash functions there, until one starts to feel good about it. See if it withstands a few obvious attacks. Then deploy the system, wait for it to get broken, and add some more padding and hashes. Repeat.

Clearly, this approach leaves much to be desired, as the above attacks on RSA clearly demonstrate. Even if the cryptosystem is built out of "cryptographically strong" components (good hash functions, hard-to-factor numbers, etc.), these components may interact in some hard-to-predict ways that allow an attacker to break the cryptosystem.

6.2 The *reductionist* approach

This is the preferred approach of modern, mathematical cryptography. Here, one shows with mathematical rigor that any attacker that can break the cryptosystem can be transformed into an efficient program to solve the underlying well-studied problem (e.g., factoring large numbers) that is widely believed to be very hard. Turning this logic around: if the "hardness assumption" is correct as presumed, the cryptosystem is secure.

This approach is about the best we can do. If we can prove security in this way, then we essentially rule out all possible shortcuts, even ones *we have not yet even imagined.* The only way to attack the cryptosystem is a full-frontal attack on the underlying hard problem. Period.

6.3 Magic hash functions

Not surprisingly, designing cryptosystems and proving them secure in this way is no easy task, especially if one wants to have a *practical* cryptosystem.

To make this task more manageable, Bellare and Rogaway use the notion of a "magic hash function," as discussed above (see [4] for a more complete discussion). The result of this approach is a reductionist proof in the above sense, but the proof is only valid in a "parallel universe" where "magic hash functions" exist—they *do not* exist in the "real world" of computation (see [8]). We stress that the existence of "magic hash functions" is not a "hardness assumption," like factoring large numbers; they simply do not exist. Rather, they are a rough-and-ready heuristic, much like assuming the earth is flat, and that there is no wind resistance.

To analyze a protocol using magic hash functions[2] one replaces a real-world cryptographic hash function by a *black box* that when queried outputs a *random bit string*, subject to the restriction that it always outputs the same value on the same input. Having made this replacement, one then gives a reductionist security argument as above. Replacing a hash function by a black box somehow captures our intuition that the output of a hash function is just some "random junk" that an adversary cannot make any sense out of.

Perhaps the right way to view a proof of security in the magic hash function model is as a proof of security against a *restricted class* of adversaries that *don't care* if the hash function really is a black box. One can easily imagine that there could be attacks of this type, and such a proof would rule out all such attacks.

Of course, there may be other types of attacks that somehow exploit the specific characteristics of the actual hash function, and these are not ruled out by a magic hash function proof of security.

6.4 Practical vs. provably secure

When Dolev, Dwork, and Naor formulated the notion of chosen-ciphertext security in 1991, they also gave a "proof of concept"; that is, they presented a particular cryptosystem and proved it secure under a standard hardness assumption. Unfortunately, their scheme is completely impractical.[3] A back-of the-envelope calculation indicates that an encryption of a single, short message would be giga-bytes in length, and would require trillions of operations on long numbers. Perhaps their scheme can be optimized somewhat, but it seems likely that it will remain hopelessly impractical.

Besides Bellare and Rogaway's OAEP, there have been several other proposals for practical cryptosystems that attempt to provide security against chosen ciphertext attack [11, 24, 17, 22]. However, none of these schemes have been proven secure, and some have been broken [15].

Thus, an obvious goal is to find a *practical* cryptosystem that is *provably*

[2]The official term is *random oracle model.*

[3]Their scheme is "polynomial time," which is a mathematician's definition of "practical," but it has little relevance to "real world" practicality.

secure against chosen ciphertext attack. This goal was first achieved with the Cramer-Shoup cryptosystem.

We will describe this system shortly, but to set the stage, we discuss a simpler cryptosystem proposed by ElGamal [14] on which the new cryptosystem is based.

7 The ElGamal Cryptosystem

We now describe the ElGamal cryptosystem, on which the newer system of Cramer and Shoup is based. There are many variations of this scheme, and we choose one that is easy to describe, but certainly not the most efficient in a computational sense.

We have to set the stage, which is a bit technical.

This scheme works with a large prime p, and as with RSA, works with the set

$$\mathbf{Z}_p = \{0, \ldots, p-1\},$$

performing arithmetic operations "mod p", just as RSA did "mod n." For technical reasons, p should be of the form

$$p = 2q + 1,$$

where q is also a prime. Such primes can be easily constructed.

For a given number a not divisible by p, its *order* mod p is defined to be the smallest positive integer x such that $a^x = 1 \bmod p$. Such an x always exists, and in fact, because of the special form of $p = 2q+1$, x is either 1, 2, q, or $2q$.

To define the ElGamal encryption scheme, we need to select a number g in $\mathbf{Z}_p$ whose order is q. That is, $g^q = 1 \bmod p$, and this holds for no smaller power. Having done so (which is computationally easy), it turns out that the sequence of numbers

$$1, g, g^2, \ldots, g^{q-1},$$

reduced mod p, contains no duplicates. So we define G to be this set of numbers.

G has several nice properties.

First, whenever we multiply or divide elements in G (mod p), we get back another element in G. Technically speaking, G is a *group*.

Second, whenever we want to compute $a^x \bmod p$ for a in G, we can always reduce $x \bmod q$, and get the same result. This is very nice, because if we want to compute $a^{xy} \bmod p$, we can first reduce the product $xy \bmod q$, obtaining a much smaller number to work with.

Third, computational experience indicates that it is very difficult to "undo" the exponentiation process in G. That is, given $g^x \bmod p$, it seems very difficult to compute x. This problem is known as the *discrete logarithm problem*, and despite intensive study for a number of years, there is no good algorithm to solve it efficiently. Interestingly, the best algorithms for computing discrete logarithms are roughly as efficient as the best algorithms for factoring—which is to say, they are *not* efficient at all.

There are a couple of problems related to the discrete logarithm problem.

There is the so-called *Diffie-Hellman problem*, which arose in connection with Diffie and Hellman's key exchange protocol [12]. The problem is this: given g^x and g^y, compute g^{xy}. Clearly, if we could solve the discrete logarithm problem efficiently, then we could also solve the Diffie-Hellman problem efficiently: compute x, compute y, and then compute g^{xy}. This problem is potentially easier to solve than the discrete logarithm problem, but currently, all evidence suggests this is not the case.

Then there is the *decisional* version of the Diffie-Hellman problem. The problem is this: given g^x, g^y, and g^z, determine if $g^z = g^{xy}$. Clearly, if we could solve the discrete logarithm problem or the Diffie-Hellman problem efficiently, we could solve this problem efficiently as well. This problem is also potentially easier than the discrete logarithm and Diffie-Hellman problems, but currently, all evidence suggests that this is not the case.

The decisional Diffie-Hellman problem underlies many cryptographic protocols—including the original Diffie-Hellman key exchange protocol.[4]

Other places where the decisional Diffie-Hellman assumption is used include [2, 3, 7, 18, 22, 23].

---

[4]It is often, and incorrectly, asserted that "breaking" the Diffie-Hellman key exchange protocol is equivalent to solving the Diffie-Hellman problem. This is not true under *any* reasonable definition of "break." This point is a bit subtle, and even seems to baffle some "card-carrying members" of the IACR (International Association for Cryptologic Research). It *is* true, however, that the hardness of the *decisional* Diffie-Hellman problem implies that breaking the Diffie-Hellman key exchange protocol is hard. There may be other hardness assumptions that imply the security of Diffie-Hellman key exchange, but it seems that the decisional Diffie-Hellman assumption is the most natural.

With this background, we can present ElGamal's encryption scheme. In this scheme, we assume that messages can be encoded as elements in G, which is easy to do, in fact.

Secret Key: random z in $\mathbf{Z}_q$

Public Key:

$$h = g^z$$

Encryption of $m \in G$**:** (u, e), where

$$u = g^r,\ e = h^r m,\ r \text{ in } \mathbf{Z}_q \text{ is random.}$$

Decryption of (u, e)**:**

$$m = e/u^z.$$

ElGamal encryption is semantically secure assuming the decisional Diffie-Hellman problem is hard. However, it is not secure against chosen-ciphertext attack. In particular, it is trivially malleable: If (u, e) encrypts m, then (u, ea) encrypts ma.

8 The Cramer-Shoup Cryptosystem

The Cramer-Shoup Cryptosystem is an extension of ElGamal. In the presentation we give here, we need a hash function H whose output can be interpreted as a number in $\mathbf{Z}_q$. It should be hard to find collisions in H. One reasonable implementation of H is to use the SHA-1 hash function. In fact, with a fairly minor increase in cost and complexity, we can eliminate H altogether.

Secret Key: random x_1, x_2, y_1, y_2, z in $\mathbf{Z}_q$

Public Key:

$$g_1, g_2 \text{ in } G \text{ (but not 1)}$$
$$c = g_1^{x_1} g_2^{x_2},\ d = g_1^{y_1} g_2^{y_2}$$
$$h = g_1^z$$

Encryption of $m \in G$**:** (u_1, u_2, e, v), where

> $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r m$, $v = c^r d^{r\alpha}$, r in $\mathbf{Z}_q$ is random, and $\alpha = H(u_1, u_2, e)$.

Decryption of (u_1, u_2, e, v)**:**

> If $v = u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2}$, where $\alpha = H(u_1, u_2, e)$
>
> then $m = e/u_1^z$
>
> else "reject"

That's it! It looks a bit strange, and it was indeed concocted just so that one could prove a theorem about it; namely, that it is secure against chosen ciphertext attack, assuming that the decisional Diffie-Hellman problem is hard, and that it is hard to find collisions in H. We repeat that one can get rid of H at a minor cost.

We won't go into the detailed mathematics of the proof here, but we can make a few comments about the intuition behind the scheme.

Notice that (u_1, e) is essentially an ElGamal encryption. The extra information, u_2 and v, is essentially a very special kind of "error detecting code." For properly constructed ciphertext, it always holds that if $u_1 = g_1^{r_1}$ and $u_2 = g_2^{r_2}$, then $r_1 = r_2$. Let's call such ciphertexts "legitimate." Now, there is nothing stopping an attacker, while performing a chosen ciphertext attack, to request the decryption of a ciphertext that is illegitimate, i.e., with $r_1 \neq r_2$. This is the point of the test

$$v \stackrel{?}{=} u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2}$$

in the decryption algorithm. This test will essentially ensure that all illegitimate ciphertexts are rejected. The group element v in the encryption acts as a "proof of legitimacy" that can be verified by the decryption algorithm. The point of the hash $\alpha = H(u_1, u_2, e)$ in the computation is to prevent proofs of legitimacy from other, legitimate ciphertexts from being "hijacked."

It turns out that by rejecting all such illegitimate ciphertexts, no information about the secret key is leaked, which effectively "neutralizes" the chosen ciphertext attack. Moreover, the "error code" information itself does not leak any useful information, as it did in Bleichenbacher's attack on RSA.

All the known constructions for chosen ciphertext secure cryptosystems (provably secure or not) are based on a similar idea of using a "legitimacy test" of some some kind. The real surprise in the Cramer-Shoup scheme was to show that such a test could be constructed quite cheaply, without giving up "provability."

How efficient is the Cramer-Shoup scheme? It is approximately twice as expensive as ElGamal, both in terms of computing time, and in terms of the size of the encryptions.

What about the cost compared to RSA? The size of the encryption and the time needed to encrypt a message are significantly more than in RSA. However, with a very careful implementation, the time to decrypt is roughly the same as for RSA. This is important since usually it is an overloaded server we are concerned about, and in many protocols, the server is mostly doing *decryptions*, not encryptions. So in that important scenario, the Cramer-Shoup scheme is quite competitive with RSA.

And in any case, the Cramer-Shoup system offers something that neither RSA nor any other practical cryptosystem does: a provable guarantee of security against chosen ciphertext attack.

We remark that there are many variations on the basic Cramer-Shoup scheme, allowing many possible security/efficiency tradeoffs. The group G could be any "cryptographically strong" group; in particular, elliptic curves could be an attractive choice.

8.1 Security of OAEP vs. Cramer-Shoup

We discuss briefly the security of OAEP versus that of the Cramer-Shoup system.

To argue in favor of OAEP, one would claim that (1) RSA is a more believable assumption than decisional Diffie-Hellman, and (2) no attacks on OAEP outside the magic hash function model are possible.

It seems quite difficult to judge claim (1): there is simply no evidence to support either claim (1) or the opposite claim that decisional Diffie-Hellman is more believable. One could argue that RSA has received somewhat more attention than decisional Diffie-Hellman, but nevertheless, all evidence to date suggests that both problems are more-or-less equally hard. Moreover, if elliptic curves are used, then the decisional Diffie-Hellman problem in fact seems substantially harder.

Although claim (2) is not entirely unreasonable, it *is* almost entirely unexamined: this is basically a claim about the hardness of a very peculiar problem that has not really been studied at all, unlike RSA or decisional Diffie-Hellman, which have already received considerable attention.

The argument in favor of Cramer-Shoup is, of course, that its security rests only on a fairly well-studied problem, and one need not worry about peculiar, almost completely unexamined problems, such as in claim (2) above.

9 Conclusion

We have tried to motivate the need for security against chosen ciphertext attack. This is the best approximation we have to the ideal of a "secure envelope," and is essential in designing secure protocols.

We have also tried to motivate the need for rigorous proofs of security. The alternative is to design systems in an *ad hoc* manner, and to simply hope for the best. As we have seen, this approach often ends in disaster, or at least, an embarrassing mess.

Practical cryptosystems that are provably secure are available, and there is very little excuse for not using them. The Cramer-Shoup cryptosystem arguably has the best security guarantee, and is reasonably practical. However, in some applications, there may be such severe engineering constraints that bar its use, in which case OAEP is arguably the next best choice.

References

[1] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology–Eurocrypt '98*, 1998.

[2] D. Beaver. Plug and play cryptography. In *Advances in Cryptology–Crypto '97*, 1997.

[3] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[4] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[5] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Crypto '94*, pages 92–111, 1994.

[6] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology–Crypto '98*, pages 1–12, 1998.

[7] R. Canetti. Toward realizing random oracles: hash functions that hide all partial information. In *Advances in Cryptology–Crypto '97*, pages 445–469, 1997.

[8] R. Canetti, O. Goldreich, and S. Halevi. The random oracle model, revisted. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[9] D. Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology–Eurocrypt '98*, pages 155–165, 1998.

[10] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology–Crypto '98*, pages 13–25, 1998.

[11] I. Damgard. Towards practical public key cryptosystems secure against chosen ciphertext attacks. In *Advances in Cryptology–Crypto '91*, pages 445–456, 1991.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22:644–654, 1976.

[13] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[14] T. El Gamal. A public key cryptosystem and signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[15] Y. Frankel and M. Yung. Cryptanalysis of immunized LL public key systems. In *Advances in Cryptology–Crypto '95*, pages 287–296, 1995.

[16] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[17] C. H. Lim and P. J. Lee. Another method for attaining security against adaptively chosen ciphertext attacks. In *Advances in Cryptology–Crypto '93*, pages 420–434, 1993.

[18] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, 1997.

[19] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.

[20] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology–Crypto '91*, pages 433–444, 1991.

[21] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, pages 120–126, 1978.

[22] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Advances in Cryptology–Eurocrypt '98*, 1998.

[23] M. Stadler. Publicly verifiable secrete sharing. In *Advances in Cryptology–Eurocrypt '96*, pages 190–199, 1996.

[24] Y. Zheng and J. Seberry. Practical approaches to attaining security against adaptively chosen ciphertext attacks. In *Advances in Cryptology–Crypto '92*, pages 292–304, 1992.