

# On pedagogically sound examples in public-key cryptography

Suan Khai Chong

Graham Farr

Laura Frost

Simon Hawley

Clayton School of Information Technology  
Monash University  
Clayton, Victoria 3800  
Australia

Email: {skcho5,gfarr,lauraf}@csse.monash.edu.au, sa\_hawley@yahoo.com.au

## Abstract

Pencil-and-paper exercises in public-key cryptography are important in learning the subject. It is desirable that a student doing such an exercise does not get the right answer by a wrong method. We therefore seek exercises that are *sound* in the sense that a student who makes one of several common errors will get a wrong answer. Such exercises are difficult to construct by hand. This paper considers how to do so automatically, and describes software developed for this purpose, covering several popular cryptosystems (RSA, Diffie-Hellman, Massey-Omura, ElGamal, Knapsack). We also introduce *diagnostic* exercises, in which all error paths lead to different answers, so that the answer given by the student may suggest the nature of their error. These too can be generated automatically by our software.

*Keywords:* sound example, diagnostic example, example generator, public-key cryptography, RSA, Diffie-Hellman, Massey-Omura, ElGamal, knapsack.

## 1 Introduction

Public-key cryptography is now taught in a large number of courses around the world, in response to the rapid development of the subject and its applications since the first papers in the area (Diffie & Hellman 1976, Rivest, Shamir & Adleman 1978). Countless textbooks, articles and lectures take students through the principles and manipulations involved in the best-known public-key cryptosystems (such as the RSA (Rivest, Shamir & Adleman 1978), ElGamal (ElGamal 1985) and Knapsack (Merkle & Hellman 1978) systems) and their relatives (such as the Diffie-Hellman key exchange scheme (Diffie & Hellman 1976) and the Massey-Omura realisation (Massey & Omura 1986) of the Shamir three-pass protocol). To learn how these systems work, it is helpful for students to have exercises on them to do by hand. These exercises will often use small numbers, so that students can see the how the various steps work and fit together without being lost in difficult computation. This is, of course, quite opposite to the demands of practical applications, where large numbers must be used in order to make cryptanalysis difficult.

If a student is to work through a particular exercise themselves, it is important that, as far as possible, they only get the right answer if they use the right method. Students may often have access to the right

answer after doing the exercise (e.g., from their lecturer, or the back of a textbook), and if their answer is the same, then their method of calculation is likely to be reinforced in their minds.

In doing public-key cryptography exercises manually, certain errors crop up repeatedly. We enumerate those that are most common in our experience in §§3–4. Of course, it is not possible to give an exhaustive list, and there is always the possibility that a student may make an unpredictable arithmetic slip or some conceptual error we have not anticipated. Nonetheless, the most common and predictable errors are worth taking some account of.

It is not altogether trivial to construct, by hand, exercises with small numbers for which common errors give a wrong answer. For example, suppose we ask a student to do RSA cryptanalysis manually for what is virtually the smallest possible example:  $p = 3$  and  $q = 5$ , so that  $n = 15$  and  $\varphi(n) = 8$ . We find that every  $d \in \mathbb{Z}_{\varphi(n)}$  is its own multiplicative inverse. Any exercise constructed with these numbers will have the unfortunate property that a student who mixes up the public and private exponents, and so encrypts instead of decrypting, will still get the right answer. The same problem occurs for  $(p, q) = (3, 7)$  or  $(5, 7)$ , and for other pairs  $(p, q)$  of small primes one may still need to choose with care as self-inverse  $d$  are reasonably numerous.

With these concerns in mind, we have studied, for the public-key cryptosystems mentioned above, and for certain kinds of exercises based on these systems, the set of exercises that are *sound* in the sense that any set of errors from a given list will lead to a wrong answer. We have constructed a program that generates sound exercises of the required sort, and enables study of the set of possible exercises.

The program may be applied not only to help the teacher construct better exercises, but as an on-line tool for use by students. A student can ask for a sound example to be generated, try to solve it by hand, enter their answer, and be informed whether their answer is right or not.

Applications of this sort suggest another, stronger property of exercises. We say that an exercise is *diagnostic* if all of the methods considered — the right one, and all the wrong ones we have allowed for — give different answers. The idea here is that the answer supplied by the student gives some evidence as to the nature of the error(s) (if any) they may have made (though the evidence is not absolutely conclusive, since a student may make unpredictable arithmetic slips or other errors we have not anticipated). Diagnostic exercises may thus enable some useful on-line feedback to the student. Our system can also generate diagnostic exercises, where they exist, and assist in study of the set of diagnostic exercises for a given cryptosystem.

The exercises we consider are generally based on

carrying out the operations of encryption and decryption in the various public-key cryptosystems. We do not cover all possible tasks that might be set for these cryptosystems, but only ones that illustrate the main manipulations involved in using the systems and for which certain kinds of errors (discussed in §3) keep coming up. The exercises generated by our system can, in any case, often be used as the basis for other exercises of slightly different character.

The concepts and software described here can also be applied to exercises using larger numbers that are designed to be solved with the aid of calculators or mathematical software. Our emphasis is on pencil-and-paper exercises, since students may have more feel for manipulating smaller numbers. Also, exercises with smaller numbers have a lower chance of being sound or diagnostic if chosen at random, so there is a clear role for automated assistance in constructing such exercises.

We assume familiarity with public-key cryptography, particularly the RSA, Diffie-Hellman, Massey-Omura, ElGamal and Knapsack systems (see, e.g., (Welsh 1988)).

This work is based on Summer Studentship projects by Chong (2003–04), Frost (2002–03) and Hawley (2002–03), and especially on Chong’s BCompSc Honours project (Chong 2003), all supervised by Farr at the School of Computer Science and Software Engineering, Monash University.

The rest of this paper is organised as follows. In the next section we give more formal definitions of the concepts of sound and diagnostic examples. In §3 we describe what seem to be the main kinds of error in the systems we consider: these occur repeatedly in exercises on all the different systems. In §4 we describe the exercise types themselves and, for each, the exact set of errors we use. In §5 we give an overview of the software, describing the main functionalities present in the system. Some conclusions and suggestions for further work are given in §6.

## 2 Definitions

In the cryptography exercises we consider, the student is given some input and must calculate the corresponding output. The inputs and outputs are numeric rather than symbolic, and usually belong to finite algebraic systems such as  $\mathbb{Z}_n$ . Exercises are usually of one of two types: encryption, where the student is given the public key and the message and must calculate the cypher (or any other information that is sent between sender and receiver); and decryption, where the student is given the public key and the cypher (and anything else that is exchanged between sender and receiver) and must work out the original message (or any other secret information shared by sender and receiver).

The aim of such a cryptography exercise is to get the student to practise carrying out some algorithm  $A_0$  for computing the desired output from the given input. The algorithm  $A_0$  may be (slightly) nondeterministic: there might be some flexibility in how the student works out the answer.

We suppose that there is also a set of alternative, incorrect algorithms  $A_1, \dots, A_k$  that each might be used, by some students, to try to solve the exercise.

An exercise  $X$  is *sound* if each of the alternative algorithms produces an incorrect answer:  $A_i(X) \neq A_0(X)$  for all  $i = 1, \dots, k$ .

An exercise  $X$  is *diagnostic* if all of the algorithms — correct or incorrect — give different answers:  $A_i(X) \neq A_j(X)$  for all  $i, j \in \{0, 1, \dots, k\}$ ,  $i \neq j$ .

We assume that the alternative algorithms  $A_1, \dots, A_k$  arise in the following way. Let  $E$  be the assumed set of errors that a student might make in executing  $A_0$ . Let  $f : \{0, 1, \dots, k\} \rightarrow 2^E$  be a bijection with  $f(0) = \emptyset$  and  $k = 2^{|E|} - 1$ . Let algorithm  $A_i$  be the algorithm obtained by trying to execute  $A_0$  but making precisely the errors from the set  $f(i) \subseteq E$  while doing so. In this way, each algorithm is identified with some subset of the error set  $E$ , with  $A_0$  being the correct algorithm in which no errors are made.

In what follows, the set of alternative algorithms will be specified by giving the error set  $E$ .

## 3 Errors

For all the systems we consider, many of the same basic errors recur, mostly involving choice of modulus, confusion between encryption and decryption, and errors in carrying out the extended Euclidean algorithm (EEA). (Recall that the EEA applied to coprime  $a$  and  $b$  produces a sequence of triples  $(\delta, x, y)$  where  $\delta = ax + by$ , culminating in  $(1, u, v)$ , where  $v \equiv b^{-1} \pmod{a}$ ). A student might choose  $u$  as the inverse instead of  $v$ ; note that  $u \equiv a^{-1} \pmod{b}$ . If  $v < 0$ , the student might ignore the sign and use  $-v$ . It is also possible that the student will take the extended Euclidean algorithm one step further, to the triple  $(0, \pm b, \mp a)$ . This is useful as a check, but a student might erroneously take one of  $\pm b$  or  $\mp a$  as the desired inverse of  $b$ , in effect treating the “0-triple” as if it were the important “1-triple”.) Most errors seem to arise through one or more of the following:

- confusion over whether to do a calculation mod  $n$  or mod  $\varphi(n)$  or mod  $n - 1$  (even students doing RSA, where  $n - 1$  has no particular role, sometimes work mod  $n - 1$  as they may recall seeing  $p - 1$  in other systems, e.g.,  $\varphi(p) = p - 1$  in the Massey-Omura system);
- using a quantity itself instead of its inverse (e.g.,  $e$  instead of  $d$  in RSA);
- taking the wrong element from the EEA calculation;
- ignoring the sign of the element taken from the EEA calculation.

## 4 Exercises and error lists

In this section, we take five systems — RSA, Diffie-Hellman, Massey-Omura, ElGamal and Knapsack — and, for each, we describe one or two types of exercise and the assumed error list for each exercise type. The exercise types are usually encryption and decryption.

Our decryption exercises usually take the position of a cryptanalyst attempting to recover the message given the cyphertext and public keys. Sometimes we use decryption exercises where the student is given part of the private key as well. There are a couple of reasons why this may be useful at times. Firstly, if that part of the private key is not given, the exercise may be of such a different kind that the errors students make are quite different to the ones we consider here. The exercise may still be worthwhile, but falls outside the scope of the present work. Secondly, pencil-and-paper exercises may use sufficiently small numbers that some particular part of the private key may be easy to guess in practice, so that the real task only begins once that part of the private key is known. (This does not mean that the lecturer necessarily gives that part of the private key when presenting the exercise to students. We treat the relevant part of the private key as known only when designing the exercise.)

For each exercise type, we state, in turn: the information assumed to be *given* to the student; what the student must *find*; the *method* (i.e.,  $A_0$ ) the student should use; and the set of possible *errors* we consider. In our software (§5), the user selects which of these errors belong to the assumed error set  $E$ . For the RSA system, we give an example to illustrate the various concepts we have introduced.

#### 4.1 RSA

##### Encryption

GIVEN: modulus  $n = pq$ , public exponent  $e \in \mathbb{Z}_{\varphi(n)}^*$ , and message  $m \in \mathbb{Z}_n$ .

FIND: cyphertext  $c = m^e \bmod n$ .

METHOD: find  $c = m^e \bmod n$ , preferably by fast modular exponentiation.

ERRORS:

- use  $\varphi(n)$  instead of  $n$  as the modulus for exponentiation;
- use  $n - 1$  instead of  $n$  as the modulus for exponentiation.

##### Decryption

GIVEN: modulus  $n = pq$ , public exponent  $e \in \mathbb{Z}_{\varphi(n)}^*$ , and cyphertext  $c \in \mathbb{Z}_n$ .

FIND:  $m = c^d \bmod n$ , where  $d = e^{-1} \bmod \varphi(n)$ .

METHOD: factorise  $n$ , to find  $p$  and  $q$ ; calculate  $\varphi(n) = (p - 1)(q - 1)$ ; find  $d = e^{-1} \bmod \varphi(n)$  using the EEA; and find  $m = c^d \bmod n$ , preferably by fast modular exponentiation.

ERRORS:

- use  $n - 1$  instead of  $\varphi(n)$  as the modulus when finding  $e^{-1}$ ;
- use  $n$  instead of  $\varphi(n)$  as the modulus when finding  $e^{-1}$ ;
- use  $e$  instead of  $d$ ;
- negate the inverse  $e^{-1}$ ;
- swap the modulus and  $d$  when computing  $d^{-1}$  (so, with modulus  $\varphi(n)$ , the student finds  $\varphi(n)^{-1} \bmod d$  instead of  $d^{-1} \bmod \varphi(n)$ );
- use  $\varphi(n)$  instead of  $n$  as the modulus for exponentiation;
- use  $n - 1$  instead of  $n$  as the modulus for exponentiation.

##### Example

Suppose we want an RSA decryption exercise with assumed error set  $E = \{(b), (c), (f)\}$ . Let  $n = 77$ , so that  $p = 7$ ,  $q = 11$ . If  $d = 17$ ,  $e = 53$ ,  $m = 12$ , and  $c = 45$ , then we could give a student the decryption exercise  $(n, e, c) = (77, 53, 45)$ . The correct method  $A_0$  would find  $d = 53^{-1} \bmod 60 = 17$  and then  $m = 45^{17} \bmod 77 = 12$ . However, this same answer is obtained if error (c) is made, but no others: putting  $d' = e = 53$  yields  $m' = 45^{d'} \bmod 77 = 12 = m$ . This example is therefore unsound. A sound example (for this error set) can be obtained by putting  $d = 17$ ,  $e = 53$ ,  $m = 6$ , and  $c = 62$ . This example is not, however, diagnostic, since the incorrect answer  $m' = 32$  is obtained either by making error (f) alone or by making errors (c) and (f). The following example may be shown to be diagnostic for  $E$ :  $n = 161$ ,  $p = 7$ ,  $q = 23$ ,  $d = 13$ ,  $e = 61$ ,  $m = 17$ ,  $c = 80$ . Diagnostic examples tend to be much harder to construct than ones that only have to be sound.

#### 4.2 Diffie-Hellman

GIVEN: prime  $p$ , primitive root  $a \in \mathbb{Z}_p$ ,  $y_1 = a^{x_1} \bmod p$ ,  $y_2 = a^{x_2} \bmod p$ .

FIND:  $K = a^{x_1 x_2} \bmod p$ .

METHOD: *either* find  $x_1$  (i.e., solve the discrete log problem  $y_1 = a^{x_1} \bmod p$ ), then form  $K = y_2^{x_1} \bmod p$ , *or* find  $x_2$ , then form  $K = y_1^{x_2} \bmod p$ .

ERRORS:

- find discrete log mod  $p - 1$  instead of mod  $p$ ;
- do exponentiation mod  $p - 1$  instead of mod  $p$ .

#### 4.3 Massey-Omura

The Massey-Omura cryptosystem follows the Shamir three-pass protocol.

##### Encryption

GIVEN: prime  $p$ ,  $x \in \mathbb{Z}_{p-1}^*$ ,  $y \in \mathbb{Z}_{p-1}^*$ ,  $m$ .

FIND:  $m^x \bmod p$ ,  $m^{xy} \bmod p$ ,  $m^y \bmod p$ .

METHOD: calculate  $m^x \bmod p$ ,  $(m^x)^y \bmod p$ , and then *either* calculate  $m^y \bmod p$  *or* find  $x^{-1} \bmod p - 1$  and calculate  $(m^{xy})^{x^{-1}} \bmod p$ . (The latter is more efficient, and is what the sender actually does when using this system. The former is sometimes done by students in exercises, and will still give the right answer if done correctly, although it cannot be done in practice since it involves the sender using information known only to the receiver.)

ERRORS:

- do exponentiation mod  $p - 1$  instead of mod  $p$ , at any stage;
- find  $x^{-1} \bmod p$  instead of mod  $p - 1$ ;
- negate the inverse  $x^{-1}$ ;
- swap the modulus and  $x$  when computing  $x^{-1}$  (so, with modulus  $p - 1$ , the student finds  $(p - 1)^{-1} \bmod x$  instead of  $x^{-1} \bmod p - 1$ );
- use  $x$  instead of  $x^{-1}$ .

##### Decryption

Here we consider the somewhat artificial situation in which the cryptanalyst is given (or has found) each party's private information, except the message, and must only recover the message.

GIVEN:  $p$ ,  $x \in \mathbb{Z}_{p-1}^*$ ,  $y \in \mathbb{Z}_{p-1}^*$ ,  $m^{xy} \bmod p$ .

FIND:  $m$ .

METHOD: *either* find  $x^{-1} \bmod p - 1$ , calculate  $m = (m^{yx})^{x^{-1}} \bmod p$ , find  $y^{-1} \bmod p - 1$  and calculate  $m = (m^y)^{y^{-1}} \bmod p$ .

ERRORS:

- find inverse mod  $p$  instead of mod  $p - 1$ ;
- use  $x$  instead of  $x^{-1}$  (or  $y$  instead of  $y^{-1}$ );
- negate the inverse;
- swap the modulus and  $x$  when computing  $x^{-1}$  (or similarly when computing  $y^{-1}$ );
- do exponentiation mod  $p - 1$  instead of mod  $p$ .

#### 4.4 ElGamal

##### Encryption

GIVEN: prime  $p$ , primitive root  $a \in \mathbb{Z}_p$ ,  $y = a^x \bmod p$ ,  $k \in \mathbb{Z}_{p-1}$ ,  $m \in \mathbb{Z}_p$ .

FIND:  $c = (a^k \bmod p, Km \bmod p)$  where  $K = y^k = a^{xk} \bmod p$ .

METHOD: find  $a^k \bmod p$ ,  $K = y^k \bmod p$ , form  $Km \bmod p$ ;  $c = (a^k, Km) \bmod p$ .

ERRORS:

- (a) do exponentiation  $a^k \bmod p - 1$  instead of  $p$ ;
- (b) do exponentiation  $y^k \bmod p - 1$  instead of  $p$ ;
- (c) do the final multiplication mod  $p - 1$  instead of mod  $p$ .

### Decryption

GIVEN:  $p, a, y = a^x \bmod p, a^k \bmod p, Km \bmod p$   
 where  $K = y^k \bmod p = a^{xk} \bmod p$ .

FIND:  $m$ .

METHOD: find  $K$  as in Diffie-Hellman, find  $K^{-1} \bmod p$ , find  $m = K^{-1}(Km) \bmod p$ .

ERRORS:

- (a) in Diffie-Hellman subproblem, to find  $K$ : find discrete log mod  $p - 1$  instead of mod  $p$ ;
- (b) in Diffie-Hellman: do exponentiation mod  $p - 1$  instead of mod  $p$ ;
- (c) find  $K^{-1} \bmod p - 1$  instead of mod  $p$ ;
- (d) use  $K$  instead of  $K^{-1}$  (i.e., as its own inverse);
- (e) negate the inverse;
- (f) swap the modulus and  $K$  when computing  $K^{-1}$ ;
- (g) do the final multiplication mod  $p - 1$  instead of mod  $p$ .

## 4.5 Knapsack

The Knapsack cryptosystem is considered insecure, but is still useful for teaching purposes. We only consider decryption exercises. Encryption, while also good for students to do, does not have much potential for the kind of errors we consider.

### Decryption

This exercise may seem somewhat artificial: the cryptanalyst is assumed to know the private multiplier  $w$ . Without  $w$ , on the face of it the student just has to solve an ordinary (nonsuperincreasing) subset sum problem manually, which involves different kinds of potential errors to the ones considered in this paper (though it is a good exercise for them to do). Also, for small manual exercises,  $w$  is often easily deduced by guessing the smallest term or two of the private superincreasing sequence and observing how they are related to the corresponding terms of the public sequence.

GIVEN:  $N, w \in \mathbb{Z}_N^*$ , public sequence  $(a_i)_{i=1}^n$ , cypher block  $c = \sum_{i=1}^n a_i m_i$ , where the  $m_i$  are the message bits.

FIND:  $m = (m_i)_{i=1}^n$ .

METHOD: find  $w^{-1} \bmod N$ , find the private superincreasing sequence  $(x_i)_{i=1}^n$  by  $x_i = w^{-1} a_i \bmod N$ , find  $w^{-1} c \bmod N$ , and find the  $m_i$  by solving the superincreasing subset sum problem  $w^{-1} c = \sum_{i=1}^n x_i m_i$ .

ERRORS:

- (a) find  $w^{-1} \bmod N - 1$  instead of mod  $N$ ;
- (b) use  $w$  instead of  $w^{-1}$ ;
- (c) negate the inverse;
- (d) swap the modulus and  $w$  when computing  $w^{-1}$ ;
- (e) do the final multiplications (to obtain the  $x_i$ ) mod  $N - 1$  instead of mod  $N$ .

## 5 Software

We have written software for generating examples with the properties discussed in §4 (Chong 2003). This section describes the main features of the system and related functionalities that give users control of the system. We begin by describing the main modes

of operation which present different ways in which the system can be used. We then briefly discuss the selection of error paths in the system (discussed in §3) and of how the information is displayed.

Source code for the software mentioned in this paper includes C programs for the example generators and Java programs for the web interface. The C files for the example generators can be obtained from:

<http://www.csse.monash.edu.au/~skcho5/CryptoTools/generators.zip>

and the Java interface files can be downloaded from:

<http://www.csse.monash.edu.au/~skcho5/CryptoTools/web.zip>

The software can be run online by using any browser that supports Java 1.1 or above<sup>1</sup> on the web-page

<http://www.csse.monash.edu.au/~skcho5/CryptoTools/Gui.html> .

### 5.1 Modes of operation

Three main modes of operation are provided in the system - Interactive mode, Random mode and Calculate mode. Each of those modes is now described in turn.

#### 5.1.1 Interactive mode

Interactive mode provides the ability to check whether an example supplied by the user is unsound, sound or diagnostic. First, the user is required to provide inputs that specify an example, typically the public key values, private key values and the message. The system then validates these inputs. For instance, for an RSA example, the value of the exponent  $e$  must be in  $\mathbb{Z}_{\varphi(n)}^*$ , and the user is warned if a self-inverse exponent is chosen.

After validation, the system can check which category the examples falls into. This mode is useful as a quick check for a manually generated example, for instance, an exercise in a textbook. However, for automatic generation of examples, random mode should be used instead.

#### 5.1.2 Random mode

Random mode is the mode by which the system performs its main function: automatic generation of random sound or diagnostic examples. At each iteration, random inputs are chosen. These must be small, and must satisfy certain validity tests (as mentioned in §5.1.1). The inputs together give a random example, from which the set of paths is constructed. On comparison of the paths, the system determines if the example is sound or diagnostic. If the example generated is not of the desired type, another iteration is attempted, and so on, until an example of the desired type is found.

Other supplementary modes such as no-filtering and target mode can be used with random mode. No-filtering mode removes the restriction on input sizes, so the example generated might use numbers that are too large for pen and paper exercises (though might still be suitable for exercises using calculators or mathematical software). It also increases the chance of a generated example being sound or diagnostic. Target mode makes the system randomly generate examples with user specified target values for some of the inputs. (The actual selected input will be within 5% of the specified target.)

<sup>1</sup>Support for Macintosh browsers is currently limited to Netscape versions only.

### 5.1.3 Probability calculation mode

Probability mode enables the user to study how common sound or diagnostic examples are, among all possible examples. For each input, the program either takes a value from the user or loops over all possible values. The proportions of sound and diagnostic examples are tabulated and the probabilities of these example types are shown. This capability allows us to study what kinds of inputs give a higher proportion of sound or diagnostic examples and also, the minimum input sizes for constructing sound or diagnostic examples.

### 5.2 Path selection

Path selection allows users to choose, from a list of possible errors (being just those error lists given in §4), which errors are to be included in the assumed error set  $E$ . This reflects the observation that some errors are more likely than others and that the error set may need to be adapted to the different needs of different groups of students.

### 5.3 Path display

In our implementation, a *path* is a sequence of numbers obtained by carrying out the successive steps of an algorithm, where path  $i$  is the path obtained from algorithm  $A_i$  (refer §2). A collection of paths gives us a *path table* where the top row of the table will always be the correct path and the remaining rows account for all other paths corresponding to all the subsets of  $E$ . An alternative path display is in the form of a *path tree* where the paths displayed are grouped by similar error subsets, which enables the user to trace the consequences of errors. The program allows the user to choose either a path table or a path tree display.

### 5.4 Using the software

The programs can be run with a variety of command-line options. Full details may be found in the man pages or in our technical report (Chong, Farr, Frost & Hawley 2004). We briefly describe the operation of the program `rsa`, which generates examples for the RSA system.

This program may be run in interactive mode using the command

```
rsa -e
```

It outputs a list of the available primes and prompts the user for  $p$  and  $q$ . Subsequent interaction allows the user to choose the other numbers used by RSA, leading to a particular choice of public and private keys, which the user may accept or reject (and try for another). The user is advised which exponents in  $\mathbb{Z}_{\varphi(n)}^*$  are self-inverse, but is not prevented from choosing such a value. Once the problem is fully specified, the program outputs all possible error paths for the decryption problem, advising the user of whether or not the example chosen is sound. Interactive mode allows study of the error paths of any example, whether sound or unsound, but does not generate examples for the user.

Random mode is the main mode of the program. Suppose the user wants a randomly generated sound example with  $n \simeq 77$ , using the same error set as in our example in §4.1:  $E = \{(b), (c), (f)\}$ . Then the user enters

```
rsa -g -t 77 -s0110010
```

The successive bits in the `-s` option are used to switch on, or off, the corresponding errors from (a)–(g) in §4.1. The program might then randomly choose the sound example given in §4.1:  $p = 7$ ,  $q = 11$ ,  $d = 17$ ,

$e = 53$ ,  $m = 6$ ,  $c = 62$ . In this case the student would be given the public key  $(n, e) = (77, 53)$  and the cyphertext  $c = 62$ , and asked to find the message  $m$ . The program's output includes a table giving the results of all error paths under our assumed error set  $E$ :

Path	d	phi_n	mod	m'
1	17	60	77	6
2	17	60	60	32
3	16	77	77	15
4	16	77	60	16
5	53	-	77	13
6	53	-	60	32

The first column here gives the path number. The second gives the value of  $d$  used. This corresponds to which, if any, of errors (b) and (c) are made, with  $d = 17, 16$  or  $53$  according as neither, (b) only, or (c) only is made. Note that errors (b) and (c) will not both be made, and that this column is not affected by whether or not error (f) is made. The third column gives the modulus used for finding  $d = e^{-1}$ , which should be  $\varphi(n) = 60$ , but will be  $n = 77$  if error (b) is made, and is inapplicable if error (c) is made. The fourth column gives the modulus used for finding  $m = c^d$ , which should be  $n = 77$ , but will be  $\varphi(n) = 60$  if error (f) is made. The fifth column gives the message found.

Path 1 in the above table is the correct path, and the remaining paths correspond to different subsets of the error set:

Path	Errors made
1	none
2	(f)
3	(b)
4	(b), (f)
5	(c)
6	(c), (f)

The output concludes with a brief remark that this example is not diagnostic and a summary of the amount of searching done before this example was found.

Generation of random diagnostic examples works similarly, with option `-d` instead of `-g`. However, it may be necessary to use larger values of  $n$ , or smaller sets of possible errors, otherwise it may be too slow, due to the rarity of such examples.

Probability calculation mode can be employed using

```
rsa -c
```

The user is prompted successively for values of  $n$ ,  $e$  and  $m$ . For each of these, the user may enter either a number or the character 'r'. If the latter, the program will examine all possible values from some appropriate range. Once all three entries have been made, the program examines all possible examples with the given values, or ranges of values, for  $n$ ,  $e$  and  $m$ , and determines the numbers of these examples that are sound or diagnostic. By default, all errors in  $E$  are permitted. If the user wants a more restricted set, then these can be specified as above. For example, to use the error set  $E = \{(b), (c), (f)\}$ , the user enters

```
rsa -c -s0110010
```

Suppose the user enters  $n = 77$  and then enters 'r' for both  $e$  and  $c$ . The program reports that it examined 159 different values of  $m$  and 40 different values of  $e$  (equivalently, of  $d$ ), and that of the examples thus determined, 1950 were unsound, 4410 were sound, and 2220 were diagnostic, giving probabilities of about 0.31, 0.69 and 0.35 respectively.

The programs may also be run using the web interface mentioned in §5.

## 6 Conclusions

We have investigated pedagogically sound examples in public-key cryptography and constructed software to generate and study such examples. Our tools aid the task of constructing good exercises for students by automatically generating examples that are sound or diagnostic, where with the absence of such tools, arbitrarily generated examples have a high chance of being unsound.

Future extensions to the tools could include creating an error feedback learning tool for students. The learning tool could use the ideas and software detailed in this report to give effective feedback on errors that students might make. The tool could also be extended to collect answers submitted by students and so carry out a more systematic study of errors that students make.

Some errors are more likely than others. If some estimates of the probabilities of the various errors were known (perhaps from the work envisaged at the end of the previous paragraph), it would be possible to generate exercises for which the probability of getting the right answer by a wrong path is bounded above by some small positive constant.

The ideas of this paper and their implementation could also be applied to other cryptosystems and possibly other teaching problems.

## References

- Chong, S. K. (2003), Cryptographic teaching tools, BCompSc Honours, Monash University, Clayton, Australia.
- Chong, S. K., Farr, G. E., Frost, L., & Hawley, S. (2004), Pedagogically sound examples in public-key cryptography, Technical Report No. 2004/155, School of Computer Science and Software Engineering, Monash University.
- Diffie, W. & Hellman, M. E. (1976), 'New directions in cryptography', *IEEE Trans. Inform. Theory* **IT-22** (6) 644–654.
- ElGamal, T. (1985), 'A public key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Trans. Inform. Theory* **IT-31** (4) 469–472.
- Massey, J. L. & Omura, J. K. (1986), Method and apparatus for maintaining the privacy of digital messages conveyed by public transmission, U.S. Patent number 4,567,600.
- Merkle, R. C. & Hellman M. E. (1978), 'Hiding information and signatures in trapdoor knapsacks', *IEEE Trans. Inform. Theory* **IT-24** (5) 525–530.
- Rivest, R. L., Shamir, A., & Adleman, L. M. (1978), 'A method for obtaining digital signatures and public-key cryptosystems', *Communications of the ACM* **21** (2) 120–126.
- Welsh, D. (1988), *Codes and Cryptography*, Oxford.