TECP – Tutorial Environment for Cryptographic Protocols

Jelena Zaitseva, Jan Willemson, Jaanus Pöial
Department of Computer Science, University of Tartu, Estonia
e-mail: `jellen@ut.ee, jan@ut.ee, jaanus@ut.ee`

Abstract

The availability of educational cryptography software is insufficient nowadays, especially for public key cryptography. We have developed a new software tool for teaching public key cryptography based on modular arithmetic. This tool is a tutorial environment allowing stepwise construction of public key cryptography protocols and demonstration of their work. It enables visualization of protocols (with values of secret parameters and intermediate results), allows adding/removing communicating parties, allows adding/editing/removing of arbitrary parameters (with dynamic recalculation of dependent parameters), handles number-theoretic and cryptographic primitives (modular arithmetic, prime numbers, generators, hash functions, etc.), allows working with gigantic integers, saves/loads constructed protocols. By means of these features the user is provided with flexible and configurable tutorial environment.

Keywords: tutorial environment, public key cryptography, modular arithmetic, visualization

1 Introduction

Since the importance of cryptography has lately greatly increased, it is very important to offer corresponding teaching courses for specialists in this area.

There is almost no educational software that would help to study this discipline. The only resource the authors managed to find was [6]. It includes only nine educational programs concerning symmetric Caesar cipher, block ciphers DES, Blowfish, IDEA and ElGamal public key encryption algorithm. It is also disappointing that there is much less educational software available explaining public key cryptography than the software explaining symmetric algorithms. This specific part of cryptography however should be available with all its nuances. Experience has shown that some students initially cannot understand how it is possible to code some information by one key and decode it by another one.

It is easy to conclude the necessity of public key cryptography teaching software. Such a software must explain the idea of public key cryptography, discover nuances and assist in understanding public key cryptography techniques. It will hopefully accelerate learning of this discipline.

The rest of the paper is organized as follows: section 2 describes requirements to the tutorial environment, section 3 gives a brief acquaintance with possibilities of TECP and describes the representation of cryptographic protocols, section 4 describes the possibilities of using TECP, section 5 gives a short description of implementation of TECP, section 6 – Conclusions and Further Work.

2 Problem Formulation

First, we analyzed primitives needed for PKC [16]. We had in mind the following protocols: Diffie-Hellman key exchange algorithm [2], RSA signatures and encryption schemes [12], Rabin public-key encryption scheme [13], ElGamal signature and encryption schemes [3, 4], DSA [10], Chaum's blind signature scheme [1].

The analysis of the cryptographic schemes shows that the tutorial environment must be able to perform the following mathematical operations:

- enable visualization of protocols, including values of secret parameters and intermediate results (all values can be arbitrary large),
- allow adding/removing communicating parties,
- allow adding/editing/sending/removing arbitrary parameters,
- handle the next number-theoretic and cryptographic primitives:
  - calculation of $a \bmod b$, $a - b$, $a + b$, $a \cdot b$, a/b, a^b,
  - calculation of $a^b \bmod n$ (-1 can also be a value of b),

- calculation of $\gcd(a, b)$,
- calculation of hash value of m,
- generation (and verification) of the prime numbers,
- generation (and verification) of a number from $\mathbb{Z}_n$ ($\mathbb{Z}_n^*$),
- generation (and verification) of a generator of $\mathbb{Z}_n^*$, provided n is a *safe prime number*, i.e. $n = 2 \cdot a + 1$, where a is a prime number. We require such a condition to be able to generate and verify efficiently a generator of $\mathbb{Z}_n^*$,
- generation (and verification) of a number congruent to $a \bmod b$,

where a, b, n and m are some positive integers.

3 Overview of Tutorial Environment

The tutorial program – TECP – is a visual environment for creation and manipulation of cryptographic protocols based on modular arithmetic (a good overview of such protocols can be found in [9, 14]). Its main part is the workfield where protocols can be created and visualized.

Protocols are viewed as sequence diagrams. By means of such diagrams the communicating parties that commit protocol steps are represented. The possible steps are generation of keys, calculations of auxiliary parameters and data transmission.

The environment has a set of tools for operating on different components of protocols, i.e. on communicating parties and protocol steps (protocol variables and data transmissions) [16]. At any time all protocol parameters can be changed by the user on the fly.

3.1 Representation of Protocol Components

Figure 1 presents a screenshot of the tutorial environment.

Parties. There are three types of parties: a regular party, an eavesdropper and a public authority.

- *Regular party* can create and calculate different protocol variables, and transmit data to any other party.

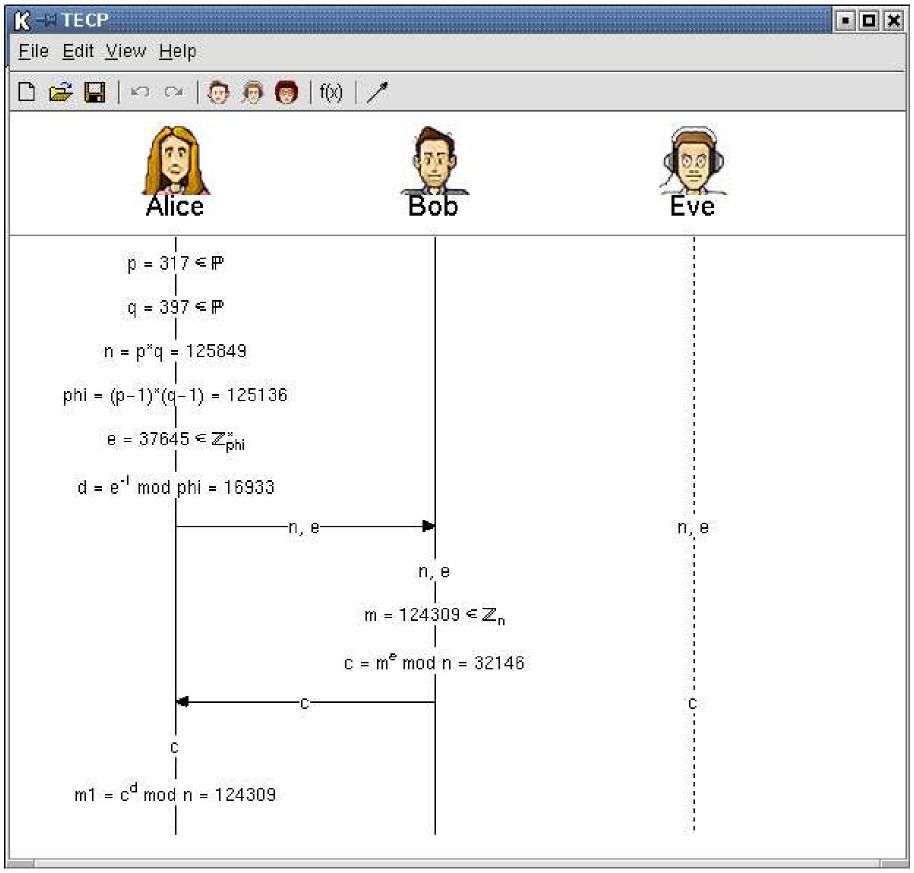

TECP
File Edit View Help
f(x)
Alice
Bob
Eve
$p = 317 \in \mathbb{P}$
$q = 397 \in \mathbb{P}$
$n = p*q = 125849$
$phi = (p-1)*(q-1) = 125136$
$e = 37645 \in \mathbb{Z}^*_{phi}$
$d = e^{-1} \bmod phi = 16933$
n, e
n, e
n, e
$m = 124309 \in \mathbb{Z}_n$
$c = m^e \bmod n = 32146$
c
c
c
$m1 = c^d \bmod n = 124309$

Figure 1: A Screenshot of the Tutorial Environment

- *Eavesdropper* is a regular party with a possibility to know all data transferred from one party to another, even if the eavesdropper is not a recipient. It can create and calculate different protocol variables, and transmit data to any other party.
- *Public authority* is a regular party able to share all received data with all other existing parties. It can create and calculate different protocol variables, and transmit data to any other party as well.

Protocol Variables. A created variable appears under a creating party as the next step of the protocol. If the value of a variable is more than 15 symbols long, only the first 5 and the last 5 symbols of the value of protocol variable will be displayed with dots in between. In this case, value of a variable can be seen by its fly-by hint or in the special window ('All Values'), which contains information about the values of all protocol variables and all the parties containing them.

Data Transmissions. Each data transmission is represented by an arrow from a sending party to a recipient with transmitted protocol variables on/above it.

4 Usage of Tutorial Environment

In this section, considerations on how the software can be used during the studying process are presented.

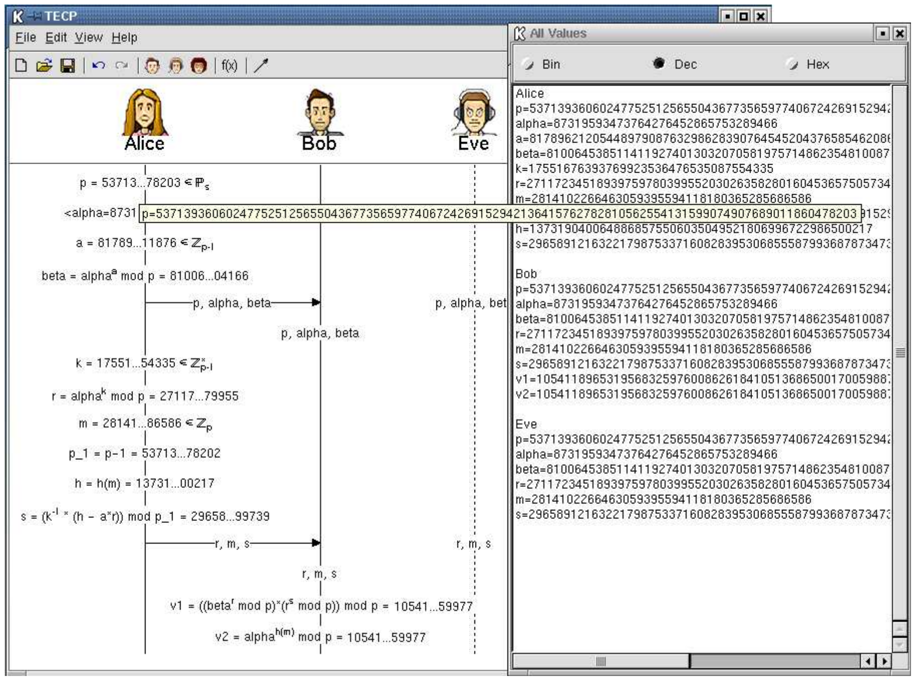
Getting to understand communication protocols. The first step students can take for understanding the idea of a protocol is to construct one using the tutorial environment.

Before constructing a protocol student often does not understand basic principles by which the protocol works, in spite of being familiar with the theoretical side of this theme. During the modeling process of the protocol comes understanding of its structure and working principles.

Visualization of numerical values of all parameters. The software enables visualization of numerical values of all parameters (protocol variables). It is considered to be significant for understanding the mechanism of the work of a protocol.

Usually, when discussing protocols at lectures using blackboard-and-chalk presentations, actual values of protocol parameters including hash values are not displayed to students due to their lengthy nature.

The tutorial environment gives convenient tools to illustrate operations with large integers including hash operations. Students can see the value of hash function themselves (Figure 2), and perform different operations with it.



```
TECP
File  Edit  View  Help
f(x)

Alice                Bob                Eve

p = 53713...78203 ∈ ℙs
<alpha=8731 p=5371393606024775251256550436773565977406724269152942136415762782810562554131599074907689011860478203
a = 81789...11876 ∈ ℤp-1
beta = alpha^a mod p = 81006...04166
        p, alpha, beta ──────▶                p, alpha, bet
                        p, alpha, beta
k = 17551...54335 ∈ ℤ*p-1
r = alpha^k mod p = 27117...79955
m = 28141...86586 ∈ ℤp
p_1 = p-1 = 53713...78202
h = h(m) = 13731...00217
s = (k^-1 * (h - a*r)) mod p_1 = 29658...99739
        r, m, s ──────▶                       r, m, s
                        r, m, s
v1 = ((beta^r mod p)*(r^s mod p)) mod p = 10541...59977
v2 = alpha^h(m) mod p = 10541...59977
```

```
All Values
Bin    ● Dec    Hex

Alice
p=53713936060247752512565504367735659774067242691529
alpha=873195934737642764528657532894 66
a=81789621205448979087632986283907645452043765854620
beta=81006453851141192740130320705819757148623548100087
k=17551676393769923536476535087554335
r=27117234518939759780399552030263582801604536575057 34
m=28141022664630593955941181803652856865 86
h=1373190400648868575506035049521806996722986500217
s=29658912163221798753371608283953068555879936878734 7

Bob
p=53713936060247752512565504367735659774067242691529
alpha=873195934737642764528657532894 66
beta=81006453851141192740130320705819757148623548100087
r=27117234518939759780399552030263582801604536575057 34
m=28141022664630593955941181803652856865 86
s=29658912163221798753371608283953068555879936878734 7
v1=1054118965319568325976008626184105136865001700598 8
v2=1054118965319568325976008626184105136865001700598 8

Eve
p=53713936060247752512565504367735659774067242691529
alpha=873195934737642764528657532894 66
beta=81006453851141192740130320705819757148623548100087
r=27117234518939759780399552030263582801604536575057 34
m=28141022664630593955941181803652856865 86
s=29658912163221798753371608283953068555879936878734 7
```

Figure 2: ElGamal Signature Scheme

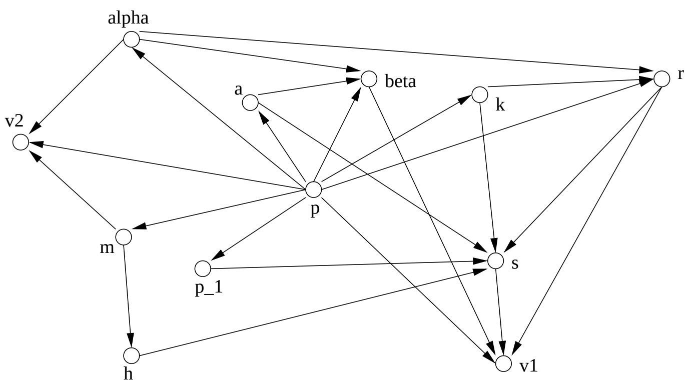


Figure 3: Dependency Graph of Protocol Variables

Possibility to change values of protocol variables. The other thing which definitely can help understanding the protocol, is the possibility to change the value of any protocol variable.

All protocol variables dependent on the edited one will be recalculated/regenerated on the fly. It becomes possible due to the dependency graph of protocol variables. In the case of ElGamal signature scheme (Figure 2) this graph is presented in Figure 3.

The change of the value of p causes the regeneration of $alpha$, a, k and m, and recalculation of $beta$, r, p_1, h, s, $v1$ and $v2$. Change of k, however, influences the values of r, s and $v1$. Change of the value of a protocol variable should convince students in the proper work of the protocol – values $v1$ and $v2$ remain equal.

Problem generation For tutors, the environment provides efficient means for generating different problem instances based on the same protocol.

E.g. a standard problem on RSA states that it is possible to recover the encrypted message if it encrypted using the public exponent $e = 3$ with three different moduli.

With TECP, an instance of this problem can be generated simply by regenerating six prime numbers.

Experimenting with protocols. Possibility of changing the protocol (addition/removal of a party, addition/change/removal of transmitted data, addition/change/removal of a protocol variable) gives the ability to see how it influences the security of a protocol.

In the simplest example of RSA encryption scheme (Figure 1), suppose, Alice sends not only n and e, but n, phi and e. In this case the user can ask Eve to calculate Alice's private key, and hence, to get to know what message Bob has sent (Figure 4).

Man-in-the-middle attacks can also be constructed using TECP.

5 Implementation

The tutorial environment was written using Borland® Kylix™ 3 Open Edition and Borland® Delphi™ 6 Personal Edition, and is available under GPL. It can be used on Linux and/or Windows machines.

Freeware package FGInt [11] and TParser 10.1 [5] were used in program implementation.

The tutorial environment can be downloaded from [15].

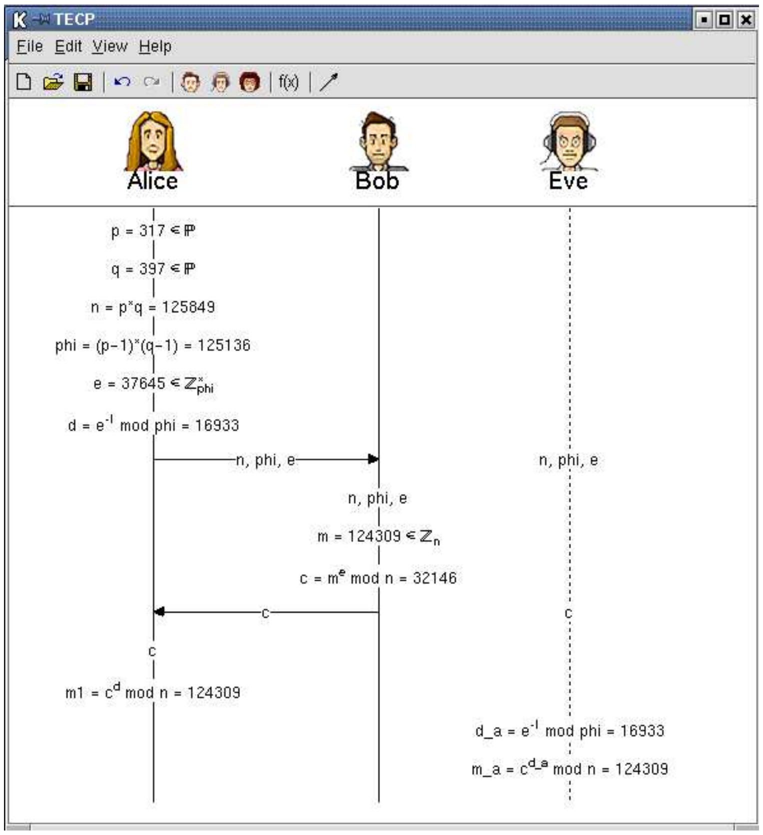


Figure 4: Successful Attack on RSA Encryption

6 Conclusion and Further Work

We have developed a new multi-platform (Linux/Windows) software tool for teaching public key cryptography based on modular arithmetic. This is an attempt to fill the gap in public key cryptography educational software.

The tool developed is a tutorial environment allowing stepwise construction of public key cryptography protocols and demonstration of their work.

TECP was used at the course 'Introduction to Cryptology' conducted of the University of Tartu in the fall term 2002. The course examination has shown that in contrast to past years examinations the amount of students who has failed it, is decreased. Although we do not have enough statistical data to judge this result, TECP proved to be a comfortable tool for a lecturer to teach and a student to experiment with protocols. It will undoubtedly be used during future courses as well. Further development of the tutorial environment can involve addition of some mathematical operations (concatenation, for example) and addition of new modules enabling constructing cryptosystems based on sparse polynomials [7] and elliptic curves [8].

Acknowledgments

The authors thank Estonian IT Foundation for support.

References

[1] Chaum, D. Blind signatures for untraceable payments. *Advances in Cryptology - Proceedengs of Crypto 82*, pages 199–203, 1983.

[2] Diffie, W. and Hellman, M.E. Multiuser Cryptographic Techniques. *Proceedings of AFIPS National Computer Conference*, pages 109–112, 1976.

[3] ElGamal, T. A Public-Key Cryptosystem and Signature Scheme Based on Discrete Logarithms. *Advances in Cryptology: Proceedings of CRYPTO 84, Springer-Verlag*, pages 10–18, 1985.

[4] ElGamal, T. A Public-Key Cryptosystem and Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, 1985.

[5] Hoffmeister, S., Flaider, A., and Schaaf, R. TParser 10.1 for Borland Delphi - a component for parsing and evaluating mathematical expressions specified at runtime. http://www.datalog.ro/delphi/parser.html, May 2003.

[6] Black Wolf's homepage. Cryptography. http://home.od.ua/˜ blackw/Crypt/crypt.html, May 2003.

[7] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Public Key Cryptosystem, August 1999.

[8] Koblitz, Neal. *Algebraic Aspects of Cryptography.* Springer Verlag, January 1998.

[9] Menezes A.J., P.C. van Oorschot, and Vanstone, S.A. *Handbook of Applied Cryptography.* CRC Press, October 1996.

[10] National Institute of Standards and Technology. Digital Signature Standard (DSS). *FIPS 186-2*, January 2000.

[11] Othman, Walied. Fast gigantic integers package. http://triade.studentenweb.org/GInt/gint.html, May 2003.

[12] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[13] Rabin, M.O. Digitalized signatures and public-key functions as intractable as factorization. *MIT Laboratory for Computer Science, Technical Report, MIT/LCS/TR-212*, 1979.

[14] Arto Salomaa. *Public-Key Cryptography.* Springer-Verlag, 2nd edition, 1996.

[15] TECP homepage. http://www.math.ut.ee/˜ jellen/TECP, June 2003.

[16] Jelena Zaitseva. *TECP - Tutorial Environment for Cryptographic Protocols.* MSc Thesis, 2003.