МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2018
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2018
Proceedings of the Forty-seventh Spring Conference
of the Union of Bulgarian Mathematicians
Borovets, April 2–6, 2018

SIMPLEST OBLIVIOUS TRANSFER PROTOCOL IN PYTHON[*]

Aleksandra Stoyanova, Lyubomir Filipov, Zlatko Varbanov

Oblivious transfer (OT) protocol is a cryptographic primitive between two parties. It can be used as a building block for any arbitrary multiparty computation protocol. Here, we are concentrating on the simplest OT type and its easiest implementation. We present an implementation in Python, a program language which can be easily learned and efficiently used for cryptography.We made tests using our implementation with three different hash functions and made comparison of the average times for their executions on different machines.

1. Introduction. Oblivious transfer (OT) is a cryptographic primitive between two parties (for example, called Alice and Bob). It is a powerful primitive in modern cryptography especially in the context of multiparty computation where two or more parties, mutually distrusting each other, want to collaborate in a secure way in order to achieve a common goal. Therefore, OT can be used as a fundamental building block for any arbitrary multiparty computation cryptographic protocol [3, 5, 6].

Oblivious transfer is a protocol by which a sender (Alice) sends some information to the receiver (Bob), but remains oblivious as to what is received. It means that Alice sends a message to Bob with some fixed probability between 0 and 1 (for example 1/2) without any information to Alice if Bob received the message or not. In other words, OT is a scheme in which Alice transfers to Bob a secret without any knowledge whether Bob received it, while Bob may or may not receive the secret, each happening with a certain probability, usually one-half.

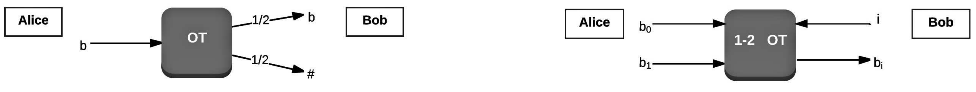
There are different versions of OT primitives, some of them can be given by their simplest versions as: **Rabin's OT** [7] and **1-out-of-2 OT** [8].

Rabin's OT can be explained in the following way: Alice chooses as input one bit b. Then, with probability 1/2, Bob gets the bit b, and nothing else (Fig. 1a).

In 1-out-of-2 OT, Alice chooses as input two bits b_0 and b_1 and Bob chooses a selection bit c and gets as output the bit b_i (Fig. 1b).

In our implementation we are concentrating on 1-out-of-2 OT, as the most powerful primitives that have led to the invention of numerous cryptographic schemes. It may conceptually be described as a black box where Alice puts in two secrets or messages, m_0 and m_1, such that Bob can only retrieve one of them, choosing one option for the bit c, while getting no information about the other. Bob is concerned that Alice should

[*]**2010 Mathematics Subject Classification:** 94A60.
Key words: Oblivious transfer, Python.



(a) Rabin's OT (b) 1-out-of-2-OT

Fig. 1. OT primitive

not know which message he retrieved. Also, 1-out-of-2 OT can be extended to 1-out-of-n OT. This 1-out-of-n OT scenario is an extension of 1-out-of-2 OT where Alice sends more messages to Bob and he can retrieve only one of them.

The 1-out-of-2 OT basic concept is presented in Figure 2, where **Alice** should not learn c and **Bob** should not learn m_{1-c}

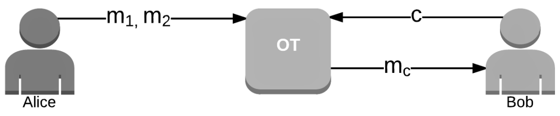


Fig. 2. OT Basic concept

2. Simplest oblivious transfer protocol implementation in Python. The Simplest Protocol for Oblivious Transfer [3] is based on Diffie-Hellman (DH) key exchange protocol [4]. Let g be a generator of a finite group. According to Diffie-Hellman protocol, Sender (Alice) and Receiver (Bob) generate random number a and b respectively. Alice computes message $A = g^a$ and Bob message $B = g^a$, and they interchange these messages. Both parties can compute $g^ab = A^b = B^a$. Using the same hash function, Alice hashes B^a, and Bob A^b, and both parties obtain the same key k. After that, when Alice sends message encrypted using the key, Bob can decrypt the message. The difference in Simplest OT is that the Receiver (Bob), chooses random c (0 or 1) and can compute different values of the message B according to c value. Therefore, in this scenario two keys are obtained on sender's (Alice's) side: k_0 and k_1 and one key k_c on receiver's side. According to this, and depending on c value, receiver can decrypt only one message received from sender (Fig. 3).

This implementation of Simplest OT uses elliptic curves. Elliptic curve used here is twisted Edwards curve [1, 2] of general form

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{1}$$

over a field $\mathbb{F}_p$, where a and d are not zeros and $a \neq d$.
Precisely, here the twisted Edwards curve **Ed25519** is used, which has the form:

$$-x^2 + y^2 = 1 - (121665/121666)x^2y^2 \tag{2}$$

over a field $\mathbb{F}_p$ where $p = 2^{255} - 19$ is prime, $a = -1$ and $d = -(121665/121666)$.

We implementated this protocol in **Python** programming language. We decided to use Python because the intentions in modern cryptography is to use Python more often and it is a favorite language for the implementation of cryptography, in particular for

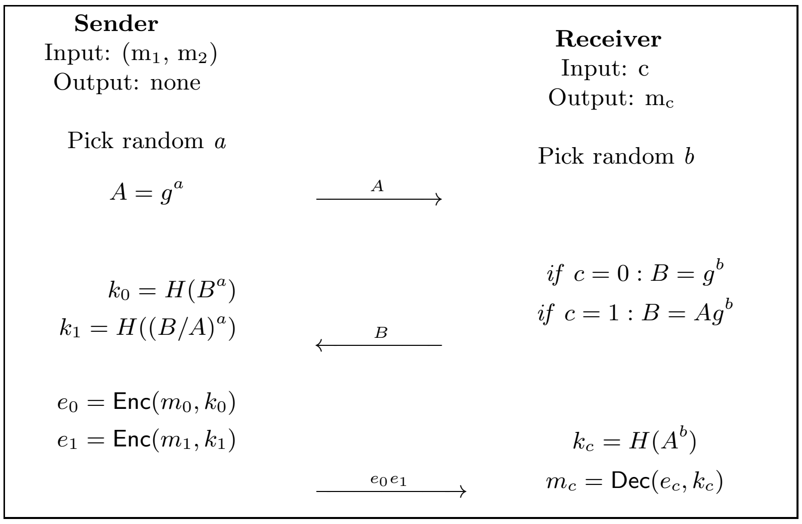


Fig. 3. Schematic view of Simplest OT protocol

analysis [9]. On the other hand, this language is easy to learn and easier to work with. There are many libraries and packages for cryptography written in Python which can be easily used. Also, Python has a huge community that provides a lot of support and is suitable for Internet of Things. That can be useful because OT as primitive can be used for secure communication in a lot of such practical applications where both parties want to remain secret in some manner and reveal as little as possible information. For the implementation we used **Python 3.6** version. We used Client-Server model for sending and receiving messages. In order to use more cryptographic functionalities, except those already included in **Python 3.6**, we used also **pycryptodome** package. In order to use **Ed25519** curve in **Python3.6** we used **joeecc** package, where this curve is implemented. The Simplest OT protocol is slightly different when **ed25529** curve is applied to it:

1. Alice samples random number $a \leftarrow \mathbb{Z}_p$, where $p = 2^{255} - 19$; Computes the value $A = a * g$, where g is generator of a group over which **Ed25519** curve is formed, and $T = a * A$; Sends message A to Bob.

2. Bob chooses random $c \in 0, 1$; Samples random $a \leftarrow \mathbb{Z}_p$; Computes message $B = c * A + b * g$; Sends message B to Alice.

3. For each $i \in 0, 1$ Alice computes $k_i = H(A, B)(a * B - i * T) \Rightarrow H(A, B)(a * B - i * a * A) \Rightarrow H(A, B)(B - i * A) * a$.

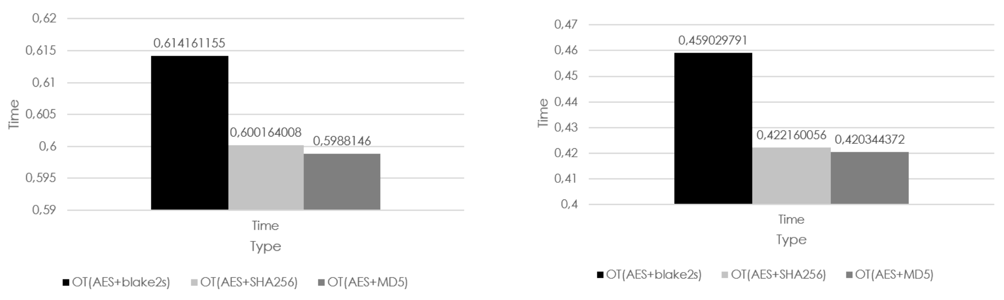
4. Bob computes $k_c = H(A, B)(B * a)$

We have used the functions **md5**, **sha256** and **blake2s** for hashing and **AES** (Advanced Encryption Standard) for encryption in our implementation.

3. Tests and results. We tested the implementation by calling 1-out-of-2 OT **100** times and **1000** times on the same machine (it means that the client and the server are on the same machine). We repeated the tests on two machines, with the following parameters:

- Ubuntu 15.04 'Vivid' (x86-64) Linux Kernel: 3.19.0-28-generic Processor: Intel Core i3-2350M CPU @ 2.30GHz x 2 Memory: 8GB
- Windows 10 pro (64 bit Operating System, x64-based processor) Intel(R) Core(TM) i7-4510U CPU @ 2.00 GHz 2.00GHz Memory: 8GB

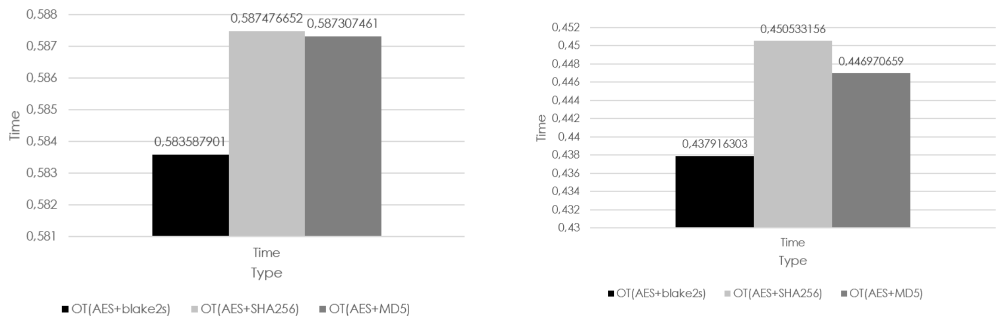
We combined **md5 with AES**, **sha256 with AES** and **blake2s with AES** and made the runtime comparison. We measured the runtime of each OT calling and according to the average times when OT is performed **100 times** we obtained the results shown on Fig. 4, and **1000 times** in Fig. 5.



(a) Machine with Ubuntu OS

(b) Machine with Windows OS

Fig. 4. Results from tests with 100 times OT on same machine



(a) Machine with Ubuntu OS

(b) Machine with Windows OS

Fig. 5. Results from the test with 1000 repetitions OT on the same machine

We made other tests by repeating OT **100 times** on **two different machines** with the same performance **connecting through network** (in this case one machine is the client and the other machine is the server). The performance of the machines is:

- Ubuntu 14.04 LTS Processor: Intel Pentium(R) Dual CPU E2180 @ 2.00GHz x 2 Memory: 1GB.

The results, obtained from these tests are shown in Fig. 6.

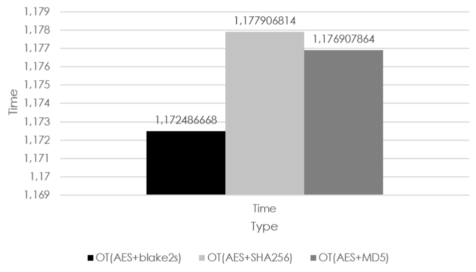


Fig. 6. Results from tests with 100 times OT on machines connected through network

4. Conclusions and future work. Our conclusions based on the test results showed that **blake2s** hash function is the fastest in the scenario when OT is called 1000 times on the same machine and in the scenario when OT is called 100 times in different machines connected through network, which means that is best suitable (compared to the other two) in real situations.

As future work we plan to include other encryption algorithms as DES or 3DES and make comparisons. Later we plan to extend the simplest protocol version to other OT version.

REFERENCES

[1] E. M. Barnard. Tutorial of Twisted Edwards Curves in Elliptic Curve Cryptography, `https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Barnard-Paper.pdf`, 2015.

[2] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, C. Peters. Twisted Edwards Curves, International Conference on Cryptology in Africa, Progress in Cryptology – AFRICACRYPT 2008, 389–405.

[3] T. Chou, C. Orlandi. The Simplest Protocol for Oblivious Transfer. In: Progress in Cryptology – LATINCRYPT 2015.(Eds K. Lauter, F. Rodriguez-Henriquez) Lecture Notes in Computer Science, vol **9230**, Springer, Cham, 2015, 40–58.

[4] W. Diffie, M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, **IT-22** (1976), 644–654.

[5] M. Naor, B. Pinkas. Computationally Secure Oblivious Transfer. *Journal of Cryptology*, **18**, Issue 1 (2005), 1–35, `https://doi.org/10.1007/s00145-004-0102-6`.

[6] A. Parakh. Oblivious Transfer based on Key Exchange. *Cryptologia*, **32** (2008), 37–44.

[7] M. O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[8] E. Shimon, O. Goldreich, A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, **28**, No 6 (1985), 637–647.

[9] Programming Languages for Cryptography, 2016, `https://cryptsec.wordpress.com/2016/01/15` (accessed 08.12.2017).

Aleksandra Stoyanova
Department of Computer Technologies and Intelligent Systems
Faculty of Computer Science
Goce Delcev University
10, Kr. Misirkov Str.
2000 Stip, Macedonia
`e-mail`: `aleksandra.stojanova@ugd.edu.mk`

Lyubomir Filipov
`e-mail`: `lyubomir.g.1991@gmail.com`
Zlatko Varbanov
`e-mail`: `zl.varbanov@uni-vt.bg`
Department of Information Technologies
Faculty of Mathematics and Informatics
St. Cyril and St. Methodius University of Veliko Turnovo
2, T.Tarnovski Str.
5000 V.Tarnovo, Bulgaria

НАЙ-ПРОСТИЯТ *OBLIVIOUS TRANSFER* ПРОТОКОЛ, РЕАЛИЗИРАН НА PYTHON

Александра Стоянова, Любомир Филипов, Златко Върбанов

Oblivious transfer (OT) протоколът е двустранен криптографски примитив, който може да се използва и като градивен елемент за протоколи за комуникация между много участници. В настоящия доклад фокусът е върху основния вид OT и най-лесният начин за неговата имплементация. Представена е имплементация на OT чрез езика Python, тъй като този програмен език е сравнително лесен за изучаване и може ефективно да се използва за криптографски цели. Чрез тази имплементация са направени тестове с три различни хеш-функции и е представено сравнение на получените резултати относно средното време за изпълнение на различни компютърни системи.