On Oblivious Transfer

Piyush Vasudha Naresh and Sivasanjai G A

Introduction

Oblivious Transfer (OT) is a fundamental protocol in the field of cryptography, enabling secure multi-party computation and serving as the backbone for various privacy-preserving applications. This cryptographic protocol facilitates a scenario where one party, the sender, transmits one of many pieces of information to another party, the receiver, without learning which piece of information was transferred.

Goals

First, in this report, we have described and implemented 1-out-of-2 Oblivious Transfer: Oblivious Transfer protocol's first and most basic variant which is based on the RSA algorithm. Then, for the final report, we intend to analyze and implement a variant of generalized Oblivious Transfer, or 1-n OT as described in the paper "The Simplest Protocol for Oblivious Transfer" by Tung Chou and Claudio Orlandi. Furthermore, we'll briefly explore the idea of universal composable security; and the refutation of UC security claims of the aforementioned paper, as described in ""The Simplest Protocol for Oblivious Transfer" Revisited" by Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial.

1-2 Oblivious Transfer: Concept and Definition

Oblivious Transfer was first introduced by Michael Rabin in the 1980s as a protocol where the sender transmits a message to the receiver with a 50% chance that the receiver receives it, and the sender remains oblivious to the reception. The concept was further refined into what is now commonly used: 1-out-of-2 Oblivious Transfer (1-2 OT).

In the standard 1-2 OT protocol, the sender has two messages, m_0 and m_1, and the receiver has a choice bit c. The receiver wishes to receive m_c without revealing c to the sender, and the sender wants to ensure that the receiver obtains only one of the messages, without learning which one was chosen.

Framework

The general setup for a 1-2 OT involves the following steps, which are secured through cryptographic means:

1. **Commitment**: The receiver commits to a choice bit c using a cryptographic commitment scheme. This step ensures that the choice cannot be changed after the fact without the sender knowing.

2. **Encryption**: The sender encrypts both messages m_0 and m_1 using keys derived in part from the receiver's commitment. The encryption method ensures that the receiver can only decrypt the message corresponding to their committed choice.

3. **Transfer and Decryption**: The receiver decrypts the message corresponding to their choice bit c using the decryption key that matches their commitment. The protocol ensures that the receiver cannot decrypt both messages.

RSA Assumptions

The security of the protocol relies on the assumption that RSA is a one-way function for randomly chosen inputs under modulus N. Given $v = y_x + k^e \mod N$, recovering k directly from v and y_x is as difficult as solving the RSA problem.

RSA based 1-2 Oblivious Transfer Protocol:

The protocol involves two parties: Alice, who holds two messages and an RSA key pair, and Bob, who holds Alice's public key and wants one of the messages. The steps are as follows:

1. Alice holds messages m_0 and m_1, and generates an RSA key pair (e, d, N).

2. Alice sends Bob two random numbers x_0 and x_1.

3. Bob chooses a random k, computes $v = y_x + k^e \mod N$, and sends v back to Alice, where y_x is the random number associated with his choice $x \in \{0, 1\}$.

4. Alice computes $k_0 = (v - x_0)^d \mod N$ and $k_1 = (v - x_1)^d \mod N$, then sends $m'_0 = m_0 + k_0$ and $m'_1 = m_1 + k_1$ to Bob.

5. Bob retrieves his chosen message $m_x = m'_x - k$.

Receiver's Privacy

Alice's inability to determine Bob's choice is ensured by the RSA encryption's one-way property. The random number k significantly alters v, making it computationally hard for Alice to deduce whether x_0 or x_1 was used in its computation.

Sender's Security

Bob learns only his chosen message m_x and obtains no information about m_{1-x}, as the random k used in the computation of v is not related to m_{1-x}.

Server Code

Global Variables

- `messages`: Contains two messages intended for secure transmission.
- `rand_msg`: Randomly generated numbers used to mask the transmission.
- `pubkey, privkey`: RSA key pair used for encryption and decryption.
- `d`: RSA private exponent.
- `N`: RSA modulus.

Server Setup

The server initializes a TCP socket and listens on port 12345. It also prepares the public key in DER format for transmission.

Functions

- **mod_exp**: Implements modular exponentiation, which is essential for RSA operations.

Main Logic

The server sends the public key and random messages to the client, receives a modified value `v`, computes responses, and sends them back to the client, ensuring secure message transfer as per the OT protocol.

Client Code

Global Variables

- `b`: Index of the desired message.
- `k`: Randomly chosen number by the client.

Client Setup

The client establishes a connection to the server via the specified port.

Functions

- **mod_exp**: Same as the server's for performing modular exponentiation.

Main Logic

The client receives the public key and random numbers, computes `v` using its secret values, and sends it to the server. It then receives and deciphers the intended message using the secret and computations performed.

A Note on Modular Exponentiation

Efficient computation of large exponentiations modulo a number is crucial due to the typically large size of the numbers involved. This note explains the method of exponentiation by squaring, focusing on its recursive implementation.

Exponentiation by Squaring Exponentiation by squaring is an efficient algorithm for raising numbers to large powers modulo a given number. This method reduces the computational complexity by exploiting the properties of exponents.

Recursive Method The recursive approach to exponentiation by squaring computes $k^e \mod N$ based on whether e is even or odd. Below, we outline the steps involved in this method.

Base Case The computation starts with the simplest case:

$$\text{If } e = 0, \text{ then } k^e \equiv 1 \mod N.$$

Recursive Reduction The exponentiation process is divided based on the parity of e:

- **If e is even:** Let $e = 2m$, then:

  $$k^e = k^{2m} = (k^m)^2.$$

  The value k^m is computed recursively, squared, and then reduced modulo N:

  $$(k^m)^2 \mod N.$$

- **If e is odd:** Let $e = 2m + 1$, then:

  $$k^e = k \times k^{2m}.$$

  First, compute k^{2m} recursively (since $2m$ is even), multiply by k, and then apply the modulus:

  $$(k \times (k^{2m} \mod N)) \mod N.$$

Efficiency The recursive approach ensures that the number of multiplicative operations is significantly reduced, with the depth of recursion being $O(\log e)$. Each recursion level involves at most a couple of multiplications and one modulus operation, providing a logarithmic reduction in computational steps compared to the linear approach. This efficiency is crucial for handling the large numbers typically used in cryptographic operations.

Extensions, and Further Work

Oblivious Transfer is crucial in applications such as secure multi-party computation, private set intersection, and zero-knowledge proofs. It has been extended in various forms, including k-out-of-n OT, where the receiver can choose k out of n messages. Think of the following scenario: a person p is hanging out with n of their friends. Now suppose, they want to compute who among them has the highest salary, without actually revealing the salaries of any particular individual. This involves securely computing a function $f(n)$ over n inputs without actually revealing n. Such problems are called, secure multi-party computation problems, and OT protocol allows for such computation.

In the final submission, it is exactly these extensions and applications we plan on analyzing and implementing.