On Oblivious Transfer

Sivasanjai G A, Piyush Vasudha Naresh

May 2024

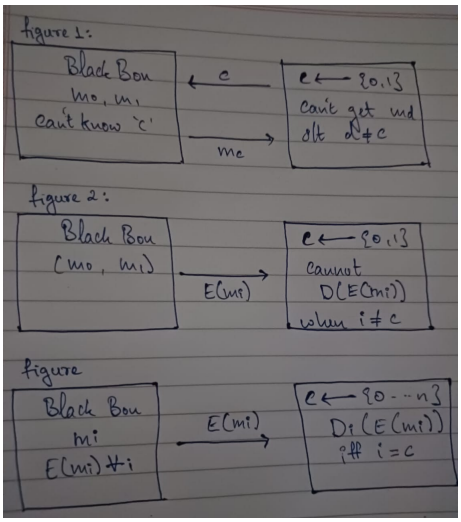
Link to github Repository

Introduction: The Black Box

Imagine a black box containing 2 messages m_0, m_1. Now suppose there exists some enquirer Q, who chooses a bit c from $\{0, 1\}$ with equal probability. The task is to get m_c corresponding to c without revealing m_i, $i \neq c$ to Q and c to the black box. The same is illustrated in Figure 1.

The task is now defined, and it is now possible to see what its successful completion would look like in cryptographic terms. Now suppose there exist encryption and decryption functions E and D with A using E to encrypt m_i and Q using D to decrypt to received ciphertext. The task at hand necessitates limiting of D such that $D(E(m_i))$ when $i \neq c$ does not work. The same has been illustrated in Figure 2.

With this image at hand, it is possible to think of our task in generalized terms beyond $\{0, 1\}$. Suppose now there exist n messages with the black box with corresponding $E(m_i)$. It is rather intuitive now to think of the function D as being D_i such that $D_i(E(m_i))$ works only when $i = c$ as can be seen in figure 3.



These descriptions capture the essence of Oblivious Transfer (OT) where information is transferred from the black box to the enquirer based on a predetermined bit with the black box and Q remaining oblivious to the choice of bit, and the rest of the messages respectively. The ability to carry out such a task holds immense value because it can be used to guarantee the security of secure multi-party computation which holds important applications in private information retrieval, and any application involving computation of functions on private information without identity compromise.

The Encryption and the Decryption:

In the previous section, properties of E and D that are necessary for OT were discussed. However, no specific encryption scheme was mentioned, as there exists a variety of different options for carrying out the same. While it may not be obvious, the description: $D_i(E(m_i))$ almost seemingly implies that OT necessitates public key encryption-like properties with a common encryption scheme and varied decryption functions for the messages.

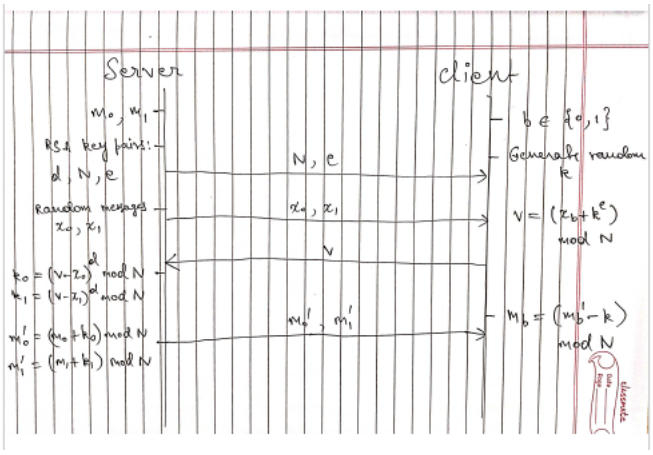
This in turn allows for the idea of using various tweaks of public key-based exchange and encryption schemes to perform oblivious transfer.

RSA-based OT

We have a server and a client program. The server has two messages m_0 and m_1. The client wants to read one of them, say m_1.
But we want this to happen such that the server does not know which message the client reads, and the client can only access one message and not the other.
So at the beginning, the server has m_0 and m_1, and the client has a choice of

$b \in \{0, 1\}$, corresponding to which message he wants to read.
In the end, the server has learnt nothing new, but the client knows m_b.

A set-up that accomplishes this can be constructed using the properties of RSA encryption.



1. Server has messages m_0 and m_1.
2. Server generates RSA key pair d, N, e and sends N and e over to client. Note: They are such that $m^{ed} \equiv m \pmod{N}$, for all $0 \leq m < n$, and given only e and N it is extremely hard to find d.
3. Client chooses $b \in \{0, 1\}$.
4. Client generates random number k.
5. Server generates random messages x_0 and x_1 and sends them over to Client.
6. Client computes $v = (x_b + k^e) \bmod N$ and sends it over to Server.
7. Server computes $k_i = (v - x_i)^d \bmod N$, for $i = 0, 1$.
8. Server computes $m_i^{'} = m_i + k_i \bmod N$, for $i = 0, 1$, and sends them over to Client.
9. Client computes $(m_b^{'} - k) \bmod N$, which gives it m_b.

Output of Server Program:

```
Connection Established. Sending public key.
Sending random messages  [72, 1]
Received v
Sending m' =  [411, 69488254048257070597728189178180272050873908583131047716026283932093776759785544811981718268330423
07385123705566512464693544247164571307919107251]
Closing connection
```

Output of Client Program:

```
Enter the index of the message you want (0 or 1): 0
Connection Established
Received public key
Received random messages
Sending v
Received m'
Requested message:  21
Trying to access the other message:  694882540482570705977281891781802720508739085831310477160262839320937767597855440
68330424625869100073851237055665124646935442471645713079191068 61
Closing connection
```

An application of RSA-based OT

We combined the RSA-based OT implementation with AES encryption to make this protocol work for arbitrary messages m_0 and m_1.

First, we encrypt m_0 and m_1 using AES encryption in EAX mode with 16 byte keys. EAX mode is an authenticated encryption mode that not only encrypts the data but also provides integrity and authenticity assurance. This mode generates an authentication tag which can be used to verify the integrity and authenticity of the data during decryption.

For each message, we return the following and send them over to the Client:

1. cipher.nonce: In EAX mode, a nonce (number used once) is automatically generated when the cipher object is created. The nonce is crucial for the decryption process, as it must be the same during both encryption and decryption. It should be stored or transmitted alongside the ciphertext and tag.

2. ciphertext: The encrypted data.

3. tag: The authentication tag is used for verifying the integrity and authenticity of the data upon decryption.

Now the two keys are appropriately converted into integers and put through the OT protocol described in the previous section. In the end, the Client will have access to the key of its choice and will be able to read the corresponding message. We also demonstrate that decryption fails for the ciphertext corresponding to the other message. This is because the Client does not have the correct key for it.

Output of Server Program:

```
Enter message 0: message 0
Enter message 1: message 1

Connection Established.

Sending RSA public key:  672239255116932538457320800352973039017268799865592934711665531211421277064463097694476617
199004361672753050743222658964545069816045766800377 9

Sending random messages  [90, 36]

Recieving v.

Sending m' =  [b'3\xec\x12\xe6\xcb\xb9\xec\xe2\x8e[\x95_\x0bLL\xa5R\x8e\x8d\xc11\xa9q\x02\r\xfe\xc0I\x0f\xe36\x18\x0
\xb2\xea\x0c\x90\x81\x01O9\xd2\xeb9\xb0/`\x8a,\xa1R\xfaM\x07\xbbH\x06\x01', b'\xac\xcd\xf8"!\xbbGw~\x1d\x80\xe7\x15\

Sending encrypted messages.

Closing connection
```

Output of Client Program:

```
Enter the index of the message you want (0 or 1): 1

Connection Established

Recieving RSA Public key.

Receiving random messages

Sending v =  184055213649853118105888790651476543687713969020965525556585804310989765567899888757368151333550717262
518690106965996955946324895096480629 4103

Recieving m' and using m_b'.

Receiving encrypted messages.
Decrypted content of message 1:  message 1

Attempting to access the other message using m_{1-b}'.
Expected failure: Decryption failed or MAC check failed for the non-chosen message.

Closing connection
```

Diffie Hellman based OT

Moving on to the next public key mechanism, the Diffie-Hellman (DH) key exchange protocol which allows two parties to jointly establish a shared secret over an insecure channel. The protocol operates in a cyclic group G of order p with a generator g. The key exchange involves the following steps:

1. Alice selects a private key $a \in Z_p$ randomly and computes $A = g^a$, which she sends to Bob.
2. Bob selects a private key $b \in Z_p$ randomly and computes $B = g^b$, which he sends to Alice.

3. Both parties compute the shared secret: Alice computes $s = B^a = g^{ab}$, and Bob computes $s = A^b = g^{ab}$.

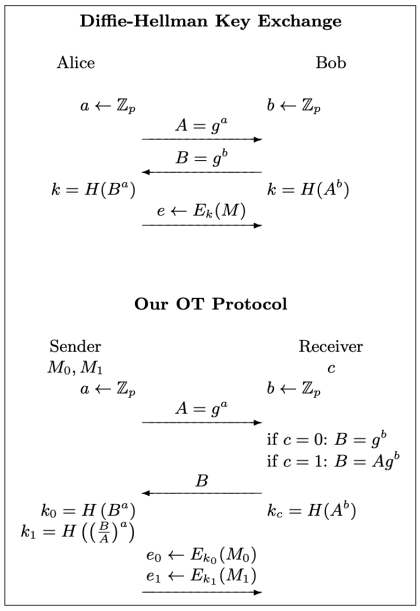
OT Protocol

This protocol extends the basic DH mechanism to enable secure 1-out-of-n oblivious transfer, leveraging the difficulty of the Discrete Logarithm Problem (DLP) in Z_p^* to ensure the protocol's security. The paper "The Simplest Protocol for Oblivious Transfer" by Tung Chou1 and Claudio Orlandi [1] outlines the following scheme:

It is possible to observe from the aforementioned description of DH-KE that Alice can also derive a different key from the value $(B/A)^a = g^{ab-a^2}$ which Bob cannot compute. With this observation, it is possible to turn the DH scheme into a random OT protocol by letting Alice play the role of the sender and Bob the role of the receiver (with choice bit c) as shown in the Figure.

1. The first message (from Alice to Bob) is left unchanged (and can be reused over multiple instances of the protocol)
2. Bob computes B as a function of his choice bit c: if c = 0 Bob computes $B = g^b$ and if $c = 1$ Bob computes $B = Ag^b$
3. Alice derives two keys $k0, k1$ from $(B)^a$ and $(B/A)^a$ respectively.
4. Bob can derive the key k_c corresponding to his choice bit from A^b, but cannot compute the other one.

Figure from Paper [1]:



Output of Code:

```
Bob's choice bit: 0
Key k0: 2ccaecdb8e4db6ba8ab06610bdb2795bc383865a97d3a8fb3e4c10675ca48751
Key k1: c4e3a50cc614e8237382fb915d9aea87f790d6184f335ea80a3fd3732515393a
Bob's derived key: 2ccaecdb8e4db6ba8ab06610bdb2795bc383865a97d3a8fb3e4c10675ca48751
Bob successfully derived k0.
```

References

[1] Chou, Tung, and Claudio Orlandi. "The Simplest Protocol for Oblivious Transfer." link