On Oblivious Transfer

Sivasanjai G A, Piyush Vasudha Naresh

May 2024

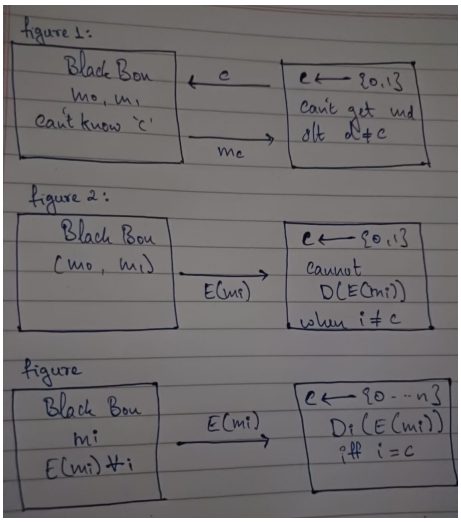
Link to github Repository

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.

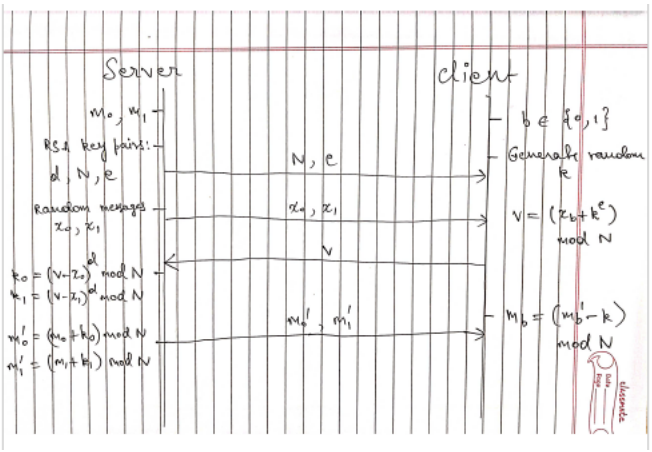
This in turn allows for the idea of using various tweaks of public key-based exchange and encryption schemes to perform oblivious transfer.

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.

Security:
The server does not know b because k is chosen randomly by the client and $v = x_b + k^e$ is sent to the server. From this, it cannot recover x_b successfully.
The client does not know x_{1-b} because of the RSA property: given only e and

N it is extremely hard to find d.
Correctness:
$(m_b^{'} - k)$ will be equal to m_b^{ed}, which is equal to m_b from the RSA property.

Output of Server Program:

```
Connection Established. Sending public key.
Sending random messages  [72, 1]
Received v
Sending m' =  [411, 69488254048257070597728189178180272050873908583131047716026283932093776759785544811981718268330420738512370556651246469354424716457130791910725
1]
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:  69488254048257070597728189178180272050873908583131047716026283932093776759785544
68330424625869100073851237055665124646935442471645713079191068610
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:  67223925511693253845732080035297303901726879986559293471166553121142127706446309769447661704
19900436167275305074322265896454506981604576680037 79

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 =  184055213649853118105888790651476543687713969020965525556585804310989765567899888757368151333550717262615
5186901069659969559463248950964806294103

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.

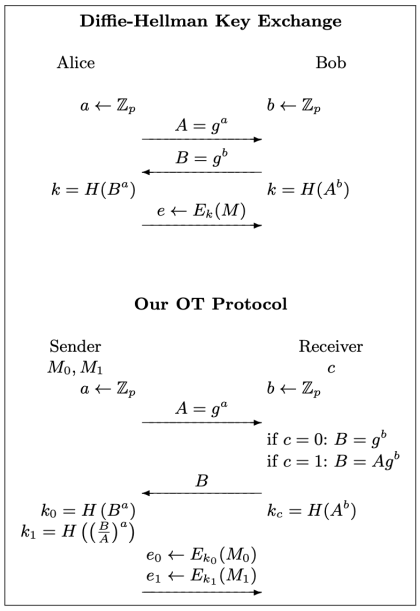
1. The first message (from Alice to Bob) is left unchanged (and can be reused over multiple instances of the protocol)

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.
```

Towards Real World Applications: 1-n Oblivious Transfer

As one ideally would see now, it is necessary to extend the previously described 1-out-of-2 Oblivious Transfer protocol to a given n in order to enable real world applications which involve, for example, secure multi-party computation. Just like in the case of $1-2$ OT, tweaks of RSA and Diffie-Hellman Key Exchange are most useful and famous for the task. The same shall now be discussed. The discussion on Diffie Hellman Based 1-n OT draws from Wen-Guey Tzeng's 2004 paper titled "Efficient 1-Out-n Oblivious Transfer Schemes". [2]

Diffie Hellman based 1-n OT:

Preliminaries

Let G be a group of prime order q, and let g and h be two generators of G. We assume that q is prime and $x \in X$ denotes that x is chosen uniformly and independently from the set X. We assume the hardness of the Decisional Diffie-Hellman (DDH) problem in G, meaning it is computationally hard to distinguish the tuple (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c) where $a, b, c \in Z_q$ are chosen independently and uniformly at random.

DDH Assumption

The DDH assumption states that given a triplet (g, g^a, g^b), it is hard to distinguish g^{ab} from a random element in G, i.e., no polynomial-time adversary can distinguish the distributions

$$D = \{(g, g^a, g^b, g^{ab}) \mid g \in G \setminus \{1\}, a, b \in Z_q\}$$

and

$$R = \{(g, g^a, g^b, g^c) \mid g \in G \setminus \{1\}, a, b, c \in Z_q\}$$

with a non-negligible advantage.

Oblivious Transfer Protocol

Assume that the system-wide parameters (g, h, G_q) are public and that the discrete logarithm $\log_h(g)$ is unknown. The following 1-out-of-n OT protocol allows a receiver R to retrieve one of the n messages from a sender S without revealing which message was chosen:

1. **Sender's input:** $m_1, m_2, \ldots, m_n \in G_q$; **Receiver's choice:** α, where $1 \leq \alpha \leq n$.
2. R sends $y = g^r h^\alpha$ to S, where $r \in Z_q$.
3. S computes and sends $c_i = (g^{k_i}, m_i(y/h^i)^{k_i})$ for each $i = 1, \ldots, n$, where $k_i \in Z_q$.
4. R computes m_α using $c_\alpha = (a, b)$ by calculating $m_\alpha = \frac{b}{a^r}$, which simplifies to:
$$m_\alpha = \frac{m_\alpha (g^r h^\alpha / h^\alpha)^{k_\alpha}}{(g^{k_\alpha})^r} = m_\alpha$$

Correctness and Efficiency

Correctness: The correctness follows from the operations:

$$b/a^r = m_\alpha (y/h^\alpha)^{k_\alpha} / (g^{k_\alpha})^r = m_\alpha.$$

Efficiency: The protocol requires two rounds, which is optimal. R sends one message y and S sends n messages c_i. R performs two modular exponentiations to compute y and m_α, and S performs $2n$ modular exponentiations.

By using fast exponentiation techniques, computational overhead can be further reduced. The security of the protocol hinges on the hardness of the DDH problem, ensuring that R learns nothing about any other m_i, for $i \neq \alpha$.

Security Proof for the Oblivious Transfer Scheme

Assume the DDH problem in a group G of prime order q is hard. Consider the OT scheme where the sender S sends messages $c_i = (g^{k_i}, m_i(y/h^i)^{k_i})$ for $i = 1, \ldots, n$ and the receiver R selects one message m_α using $y = g^r h^\alpha$.

Claim

For $i \neq \alpha$, each c_i is computationally indistinguishable from a random tuple in the form $(g, h, g^a, m(g^r h^\alpha / h^i)^a)$, under the assumption that the DDH problem is hard.

Proof

Given the protocol, we know:

$$y = g^r h^\alpha$$

For $i \neq \alpha$, c_i is:

$$c_i = (g^{k_i}, m_i(y/h^i)^{k_i})$$

Substituting y, we have:

$$c_i = (g^{k_i}, m_i(g^r h^\alpha / h^i)^{k_i}) = (g^{k_i}, m_i(g^r h^{\alpha - i})^{k_i})$$

Assumption Analysis: The DDH assumption implies it is hard to distinguish (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c) for random a, b, c. If we can link this to our case, c_i for $i \neq \alpha$ should look like a tuple from the second distribution $(g, g^{k_i}, g^r, m_i g^{r k_i} h^{(\alpha - i) k_i})$.

DDH Application: Consider the tuple $(g, g^{k_i}, g^r, g^{r k_i})$ which under DDH, is indistinguishable from (g, g^{k_i}, g^r, g^c) for some random c. Now, if we replace g^c with $m_i g^{r k_i} h^{(\alpha - i) k_i}$, unless R can solve DDH, it cannot differentiate whether the third component is formed by r and k_i or some random exponent, and thus c_i appears random.

Reducing DDH to our Setting: If there exists an adversary A that can distinguish c_i from random when $i \neq \alpha$, then A could be used to break the DDH assumption by distinguishing (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c). This reduction shows that breaking our protocol's security would imply breaking the DDH assumption, which is assumed hard.

Conclusion

Under the DDH assumption, R cannot differentiate between the encrypted m_i for $i \neq \alpha$ and a random element, ensuring that the OT protocol is secure as long as the DDH problem remains hard. This concludes that R gets no information about any m_i for $i \neq \alpha$.

1-n RSA-Based OT

The previous code for 1-2 RSA-based OT can be easily extended to perform 1-n OT. The running of the program is as follows:

1. Server has messages $m_0, ..., m_{n-1}$.
2. Server generates RSA key pair d, N, e and sends N and e over to client.
3. Client chooses $b \in \{0, ..., n-1\}$.
4. Client generates random number k.
5. Server generates random messages $x_0, ..., x_{n-1}$ 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 $i = 0, ..., n-1$.
8. Server computes $m_i^{'} = m_i + k_i \bmod N$, for $i = 0, ..., n-1$, and sends them over to Client.
9. Client computes $(m_b^{'} - k) \bmod N$, which gives it m_b.

Security:
The server does not know b because k is chosen randomly by the client and $v = x_b + k^e$ is sent to the server. From this, it cannot recover x_b successfully.
The client does not know x_c for some $c \neq b$ because of the RSA property: given only e and N it is extremely hard to find d.
Correctness:
$(m_b^{'} - k)$ will be equal to m_b^{ed}, which is equal to m_b from the RSA property.

Output of Server Program:

```
(sivasanjai) (base) sivas@Sivasanjais-MacBook-Air CSP_Jhawar % /Users/sivas/anaconda3/envs/sivasanjai/bin/python "/Users/sivas/Docume
ion/Sivasanjai Mi
gration/Desktop/CSP_Jhawar/Server_OTn_CSP_Project.py"
Enter the number of messages: 7
Enter message 1: 1
Enter message 2: 2
Enter message 3: 45
Enter message 4: 33
Enter message 5: 89
Enter message 6: 60
Enter message 7: 7
Connection Established. Sending public key.
Sending random messages  [50, 74, 33, 62, 75, 12, 31]
Received v
Sending m' =  [48259901888891049032696232313918444016732412012992836096401569711215096728277790041131165668789248044569823426487474218
84819472797270126487896604, 42049081923138955206432108542227138925103950187441924109706237219520323005253453018720998063351825107784507
07983554091238006359492306165085511747O, 39033456097749721899442307484462185117909658765899133074008987805848728448579111392837965171
68048199754220332749385181814151758291931266223591B5, 78414528930623027713126527675955745744363879748617465451282923579871639740259380
70334231346740306820368163700256115391782002996097714754700143176, 48748889088822439649101035584256162437942380180371429715451212125O
24425111684186651502427128543924737281685228317493804642798097028774553797377 7, 43829268492335651624667802571909075739325170339844142
19443809594646309844583348613221067211902601585830044541445040340641372809160482767452204 40, 318]
Closing connection
(sivasanjai) (base) sivas@Sivasanjais-MacBook-Air CSP_Jhawar %
```

Output of Client Program:

```
(sivasanjai) (base) sivas@Sivasanjais-MacBook-Air CSP_Jhawar % python Client_OTn_CSP_Project.py
Connection Established
Received public key
Received random messages
Enter the index of the message you want (0 to 6): 6
Sending v
Requested message:  7
Closing connection
(sivasanjai) (base) sivas@Sivasanjais-MacBook-Air CSP_Jhawar %
```

Real-world application of 1-n OT

We combine our 1-n OT code with AES encryption in EAX mode (done before) to simulate an application for secure and private sharing of songs.

The server can be thought of as a platform for users to access music they have paid for. So the user only gets access to one song. At the same time, the server must not know which song the user has accessed. This is done to ensure user privacy and perhaps eliminate targeted advertisements on the server's behalf.

All this can be achieved using 1-n RSA-based OT along with AES encryption in EAX mode as follows:
The server maintains a larger repository of songs. The link to each song is encrypted using AES in EAX mode, and the corresponding encryptions are sent to the client. Now 1-n OT is performed on the keys. So in the end the client has access to exactly one key, which it can use to decrypt the corresponding link and access the song. No other key is learnt by the client and the server does not know which song the client has accessed.

Output of Server Program:

```
Connection Established.

Sending RSA public key:  753024950465251590956695461969456722247926815346147193593433702248257160690408816226299051594218194173
465816845711217752761028670271004384702032555286587l

Sending random messages  [51, 30, 72, 80, 39]

Recieving v.

Sending m' =  [b'\x8bp\xa7\x04)\x93\xc0=\xe2\xa0\x9b\xdb\x0e\x1eD\xbe\xaa\xdc9\xcb\x96\\\xb6FKy\x0fe!W\xf7\x0c\x84WE\xdd\xef\x0
7\xf7Z\x014\xcf\n\xa8\xf2}\x18Y\x7f\xf9=i3\xd2\xee7\xf3W\xb9~SY^', b'\x7f=\xf8\x1cy\xd0\x06^\xe69\xc1\xfa\x03\x9dM#t\x82\x8e\xb
eKrn\x03X9\xa7\x92\xc7\x81\xffO\x87\x96<\x0b\xbc1\xa1\xb4~\x93\xae\x9a\x82y;@\\3$\xe5U\x9a09\xa1t\x99a\xd1\x86\xb7\x18', b'\x81
Q2\xe4\x93\xd3\xfd\xb4 \x90\x83\x009\xe0e\xca\x16\xbd\x00\x0eK\x04\xf90}O\r\xd5\xcd\x8esJ\xe0`\x06#6aJ\x0e\x1f\x1e9\xe3\x98\x0f
h,\x9c\x1aj~\xc0>\x0b`M9[\x8c\x86\xd95w', b'$\xf8=a\xca\x14\xd1n\x9el\xe5\x81\x02Q~\x07', b'W\x04\x15\x95\xf69\xc3Uz\xeb\xab"\x
b8\xc5\x1ed\x901S\x8e\xb9{\x0c\x87\xef\xe5Z\xde7\x16\xcf+\x04\xd9{\xd9\xd2\x11a5\xb1I4Y\x9a\xa81A-a\xd8p\xee\xd7_iT*\xcdi\xde\x
e4*|']

Sending encrypted messages.

Closing connection
```

Output of Client Program:

```
Enter the index of the message you want (0 for Believer, 1 for Cupid, 2 for Baby, 3 for Love Story, 4 for Bad Blood): 3

Connection Established

Recieving RSA Public key.

Receiving random messages

Sending v =  475774243129045758303968099774921268137874138013838240713017043963469400334035203833378768849133658482140739309046
365472730049096102752757044170681019648 6

Recieving m' and using m_b'.

Receiving encrypted messages.
Decrypted content of message 3:  https://youtu.be/8xg3vE8Ie_E?si=Ytxh8ATchaCZE8_Q

Closing connection
```

References

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

[2] Tzeng, Wen-Guey. "Efficient 1-Out-n Oblivious Transfer Schemes."link