[Oblivious and fair data trading protocol by using Blockchain]

Master's Thesis mid-semester report submission by

KOTHAKAPU SAIBABA
M.Tech in cryptology and Security
Roll No. CrS 2007.

Under the guidance of

Dr.Souradyuti Paul

Department of Electrial Engineering and Computer Science

Indian Institute of Technology(IIT), Bhilai.

Institute Supervisor

Dr.Anisur Rahaman Molla

Cryptology and Security Research Unit

Indian Statistical Institute(ISI), Kolkata.

Cryptology and Security Research Unit
Indian Statistical Institute, Kolkata.

ACKNOWLEDGEMENT

I want to start by expressing my sincere gratitude to my adviser, Dr. Souradyuti Paul, for his thorough direction, contagious encouragement, and helpful critiques required for this study effort. His guidance and expertise were very helpful to me as I conducted quality research and wrote this thesis. I am fortunate to have such a wonderful supervisor since he continually encouraged me on a personal and intellectual level.

I would like to thank my internal supervisor Dr.Anisur Rahaman Molla, for supporting me in all kinds of situations and extending his helping hand to the fullest possible.

I would like to thank Dr.Ananya Srivastava, Mohammad Sumair for helping me to understand the problem more deeply and explaining the various aspects of the problem.

Special thanks to Subra majumdar and my friend M.Shiva Kumar for clearing my doubts and in helping me by providing the required material to read.

I also wish to thank my friends Manish Kumar, Maneesha Panda, Asif Kalam for their encouragement and support.

Last but not the least I'm extremely grateful to my parents and my family members for their love and support.

K.Saibaba

M.Tech in Cryptology and Security

ISI Kolkata

August 2022

ABSTRACT

In oblivious and fair data trading protocols there will be two parties owner and buyer. The owner will be having a set of files along with the keywords associated with them and the buyer will have a search word. Buyer wants to buy a file from the owner which contains the search word without revealing his search word to the owner and similarly, owner also doesn't want to reveal his files till the buyer pays him.

In this dissertation, I have carried on the work previously done by my supervisor along with his students. They have come up with a protocol(The OFT protocol)4.1.1 to solve this problem but it had an issue. In this dissertation, we have come up with a new protocol called "The new protocol:OFT" which solves the problem.

Contents

CERTIFICATE

This is to certify that the dissertation entitled **'Oblivious and fair data trading protocol by using Blockchain'** submitted by **Kothakapu Saibaba** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Cryptology and Security** is a bonafide record of work carried out by her under our supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in our opinion, has reached the standard needed for submission.

Dr.Anisur Rahaman Molla
[Internal Supervisor]
RC Bose centre for
Cryptology and security
Indian Statistical Institute, Kolkata

Dr.Souradyuti Paul
[External Supervisor]
Department of Electrical
Engeneering and Computer Science
Indian Institute of Technology, Bhilai

Chapter 1

Introduction

Because of the Digitization across the globe, there is extensive growth in the generation and sharing of digital data. This generated data has becom crucial for everyone. So buying and selling the digital data has become a part of our regular life. During this exchange of the data, the Owner doesn't trust the Buyer, the Buyer doesn't trust the Owner. Fair exchange protocols are playing a key role in the exchange of digital goods.

In the Oblivious and fair data trading protocol, The buyer wants to buy a File that contains a specific keyword. The buyer will search with his keyword. If there is any file that contains the specific keyword. He will buy from the Owner. But the problem is Buyer doesn't want to reveal the search word to the Owner. And the owner allows the buyer to get the data only if he pays the money.

In this dissertation, we need to come up with a protocol that guarantees both Privacy as well as Fairness in the protocol. Which means that at the end of the protocol privacy of the Files, Searchword should not be broken. And either the Parties should get what they have been promised or no one should get anything.

1.1 Motivation

Oblivious Search [5] and Fair exchange of digital goods [2] are well-studied problems . In a 2 party Oblivious search protocol Owner O will have a set of files, and Buyer B will have Search word, Reward money$M$. At the end of the protocol, the Buyer should get a file that contains the search word. But the problem here is to keep the search word, and files secretly.

To solve this problem many people have worked, Boneh et al [1] . and Popa [8] et al. proposed two different solutions. But later Grubbs et al. have shown that these protocols are suffering from search word privacy: if a malicious server colludes with the owner then it leaks the information about the search word queried by the buyer, as well as the matched document. Another attempt was done by Ogata and Kurosawa who designed their protocol in peer to peer setting. But unfortunately, any of these protocols do not support trading because there is no dispute-resolving mechanism. So any of these protocols do not ensure the fairness which is required to perform the trading.

After the development of the Blockchain technology some people even try to solve this problem using the blockchain. But any of these solutions do not guarantee fairness because of a lack of dispute resolution mechanism.

So now the following questions arise.

1. Can we design an *Oblivious search with data trading protocol*.

2. If the answer to the above question is yes.
   Can we design an *Oblivious and fair data trading protocol* which guarantees both the privacy of the file as well as the keyword.
   In this dissertation the answer for the above two questions is indeed yes.

Chapter 2

Background of the Problem

To construct Oblivious and fair data trading protocol(OFT) we have mainly used the following primitives such as

1. Bilinear map
2. Hash function
3. 1-2 oblivious transfer protocol
4. Zero Knowledge proofs
5. Blockchain and Smart contracts

2.1 Bilinear map

Definition 2.1.1. Let $(G_1, .)$ and $(G_T, .)$ be the groups of prime order p. Also,let g_1 be the generator of G_1. A bilinear map is a mapping $e : G_1 \times G_1 \to G_T$ satisying the following properties:

1. Bilinear: we say that a map $e : G_1 \times G_1 \to G_T$ is bilinear, if $e({g_1}^{k_1}, {g_1}^{k_2}) = e(g_1, g_1)^{k_1 k_2}, \forall {g_1}^{k_1}, {g_1}^{k_2} \in G_1$and $\forall k_1, k_2 \in {\mathbb{Z}_p}^*$.
2. Computable: Given ${g_1}^{k_1}, {g_1}^{k_2} \in \mathbb{G}_{\mathbb{1}}$ there is a polynomial time algorithm to compute $e({g_1}^{k_1}, {g_1}^{k_2}) \in \mathbb{G}_{\mathbb{T}}$.
3. Non-Degeneracy: If g_1 is the generator of G_1 then $e(g_1, g_1)$ is a generator of $\mathbb{G}_T$.

2.2 Hash Function

Definition 2.2.1. [9] Hash function $H : \{0,1\}^* \to \{0,1\}^n$(where n is a fixed length for example n=256 or 512)is a deterministic polynomial time algorithm which takes an arbitrary length of the message as an input and a fixed n bits length of the string as an output which satisfies the following properties:

- **Collision resistant:** For all probabilistic polynomial time adversary $\mathcal{A}$, there exists a negligible function $negl$ such that:

$$Pr[Hash - coll_{\mathcal{A},H}(\lambda) = 1] \leq negl(\lambda).$$

where the game $Hash - coll_{\mathcal{A},H}(\lambda)$ is defined as follows:

1. The adversary $\mathcal{A}$ outputs a pair x and x'.
2. The output of the experiment is 1 (i.e $\mathcal{A}$ wins) if and only if $x \neq x'$, and $H(x) = H(x')$. In such a case we say that $\mathcal{A}$ has found a collision.

- **Preimage resistant**: A Hash function is said to be preimage resistance if $\forall$ efficient adversaries$\mathcal{A}$, the quantity$Pr[\mathcal{A}, H]_{win}$ is negligible. Where $Pr[\mathcal{A}, H]_{win}$ is probability that $\mathcal{A}$ wins in an attack game which is defined in the following way.

  1. For any given $h \in \{0,1\}^n$ the adversary outputs x.
  2. The adversary $\mathcal{A}$ wins if $H(x) = h$. such x is called preimage of h. The probability that adversary wins in this game is denoted by $Pr[\mathcal{A}, H]_{win}$.

2.3 1-2 oblivious transfer protocol

1-2 oblivious transfer protocol is a two-party protocol. Let's say that the two parties are Alice(Sender) and Bob(Receiver). In this protocol Alice, the sender has two messages m_0, m_1 wants to send only one message without receiver learning about the other message. Bob will choose a bit $b \in \{0,1\}$ wants to receive a message m_b without sender learning about the bit b.

Alice				**Bob**		
calculas	secret	public		public	secret	calculus
Messages to be sent	m_0, m_1					
Generate RSA key pair and send public portion to Bob	d	N, e	$\rightarrow$	N, e		receive public key
Generate two random messages		x_0, x_1	$\rightarrow$			receive x_0, x_1 publicly
					k, b	choose $b \in \{0, 1\}$ and generate random key k.
			$\leftarrow$	$v = x_b + k^e mod N$		computes a value v and send it to Alice.
One of these will equal k, but Alice does not know which	$k_0 = (v - x_0)^d mod N$ $k_1 = (v - x_1)^d mod N$					
Send both messages to Bob		$m'_0 = m_0 + k_0$ $m'_1 = m_1 + k_1$	$\rightarrow$	m'_0, m'_1		receive both messages
					$m_b = m'_b - k$	Bob decrypts the m'_b since he knows which x_b he selected earlier.

2.4 Zeroknowledge proofs

In this section we will discuss about Interactive proof systems, Zero knowledge proofs, The existence of zero knowledge proofs for NP Languages.
The goal of any proof system is to convince someone that a certain statement is true. Where a statement is an instance of a language L . So a statement consists of the tuple (x, L) or more precisely $x \in L$.

Example 2.4.1.

1. "P is a prime number".
   $p \in \{n | n \text{ is a prime number}\}$
2. "x is the discrete log of h base g"
   $x \in \{y | g^y = h\}$.
3. "a given graph G_1 is 3 colourable".
   $G_1 \in \{G | G \text{ is 3 colourable}\}$.

2.4.1 Interactive proofs

Definition 2.4.2. Interactive proof: Interactive proof system for a language L is a protocol between two parties prover P and verifier V. At the beginning of the protocol both the Prover P and verifier V are given some instance x. At the end of the protocol V either accepts or rejects the statement $x \in L$.
A useful proof system should satisfy the following properties:

1. **Completeness:** If $x \in L$, Then any honest P should convince any honest V.

2. **Soundness:** If $x \notin L$, Then any dishonest P should not be able to convince the V.

Now the question arises what type of languages have an Interactive proof systems? The simplest example is the NP types of languages.
Let $L \in NP$. Then by definition $\exists$ an efficient algorithm $M(.,.)$ suchthat

$$x \in L \Leftrightarrow \exists\ w \in \{0,1\}^{pol(|x|)} \text{ suchthat } M(x,w) = 1$$

Therefore, we can give the protocol as follows:

1. $P(x)$ will compute the witness and share it with the $V(x)$.
2. now $V(x)$ runs the algorithm M with x and w as inputs and accepts the statements $x \in L$ if $M(x,w)$ outputs 1. rejects otherwise.

So in this way we can have one shot non interactive proofs protocols for every $L \in NP$. So the question arises why do we need interactive proof systems? when we already have non interactive proof systems then why do we need interactive proofs? we need interactive proofs for following two reasons:

1. some times we want to prove the statements which are not in the NP languages. For example if we take a statement "Boolean formula ϕ does not have a satisfying assignment." In this statement the instance of the language belongs to **coNP**. These kind of statements are difficult to prove without interaction.

2. some times even though the instance of the language belongs to the **NP** if the size of the interaction is less than the size of the witness then it is better to go with the interactive proofs.

Now lets define the Interactive proofs formally.

Definition 2.4.3. Interactive proof Let L be any language. Let $< P(x), V(x) >$ be a protocol specification between a prover P and the verifier V. Then, we say that $< P(x), V(x) >$ is an interactive proof system for L if the following two properties are satisfied:

1. **Completeness:** If $x \in L$,

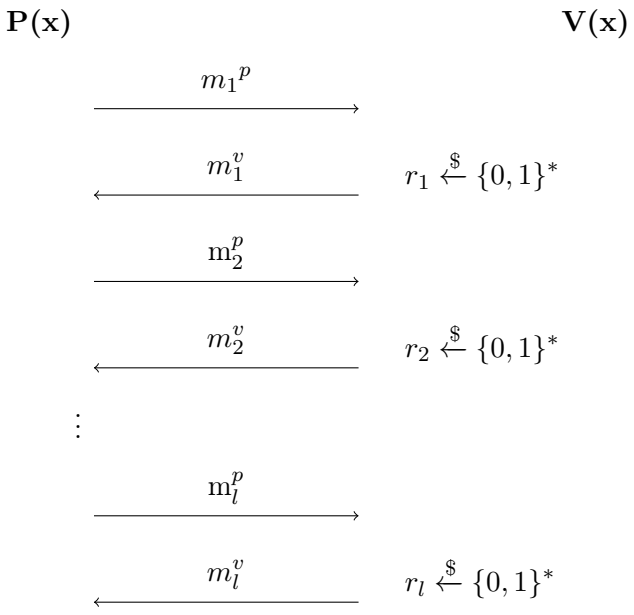
$$Pr[< P(x), V(x) >= 1] \geq 1 - negl(\lambda)$$

2. **Soundness:** If $x \notin L$,

$$Pr[< P(x), V(x) >= 1] \leq negl(\lambda)$$

2.4.2 Zeroknowledge

Till now in interactive proofs we have seen that P is proving a statement $x \in L$ by sending a witness w to the V. Now the question is can we prove a statement $x \in L$ with out revealing anything else? For example I want to prove that "I know a solution to the given suduko puzzle with out revealing the solution".



In this section we are restricting ourselves zero knowledge proofs for NP problems. There are two parties:

1. An honest prover P with an input (x, w) behaves according to the specifications of the protocol.

2. A Dishonest verifier V^* with an input x can deviate from the specifications of the protocol and tries to get the information from the prover$P$.

In this way during the protocol verifier is getting $(m_1{}^p, m_2{}^p, m_3{}^p ... m_l{}^p)$ and $(m_1{}^v, m_2{}^v, m_3{}^v ... m_l{}^v)$ and internal randomness r_i for $i \in \{1, 2...l\}$ values. From all these things verifier tries to learn additional knowledge about x.

we define the view of V^* as the random variable

$$view_{P,V^*(x)}=(m_1{}^p,m_2{}^p,m_3{}^p...m_l{}^p,r_1,r_2,r_3..r_l)$$

informally we can say that a protocol is satisfying zero knowledge property if no dishonest verifier $V*$ can get any additional information about x by looking at the $view_{P,V^*(x)}$. How do we define the knowledge? Here knowledge is defined with respect to the things we can compute efficiently.For example in a prime factorisation problem of a number N such that N is the product of two large prime numbers p,q. If we know one factor p we have the knowledge of other factor q as well because we can efficiently compute q from N and p. But where as in the case of having encryption of a file we can't say that we have knowledge of the file.
we can say that a protocol is said to be satisying zero knowledge property if whatever the knowledge the verifier is gertting from the transcripts of the protocol between prover and verifier. If verifier can get the same amount of knowledge from running the protocol between verifier and some random person(with the knowledge of the witness).

Definition 2.4.4. Zero knowledge proofs: Let $L \in NP$. Let $< P, V >$ be a protocol specification between a (possibly unbounded) prover P and a (PPT) verifier V. Then, we say that $< P, V >$ is an interactive proof system for$L$ if the following properties are satisfied:

1. **Completeness:** If $x \in L$,

$$Pr[< P(x,w), V(x) >= 1] \geq 1 - negl(\lambda)$$

2. **Soundness:** If $x \notin L$,

$$Pr[< P(x,w), V(x) >= 1] \leq negl(\lambda)$$

3. **(computational)Zero knowledge:** if $\forall V^*$,$\exists (PPT) Sim_{V^*}$ such that $\forall x \in L$,

$$View[< P(x,w) \Leftrightarrow V^*(x) >] \approx Sim_{V^*}(x)$$

Zeroknowledge proofs for NP

In this section we will discuss the existence of zero knowledge proofs for NP language. First we will see the existence of a zero knowledge proofs for NP complete language and by definition of NP complete language we can reduce any NP language to the Np complete language so the existence of zero knowledge proof for an NP complete language implies the existence of zero knowledge proof for all NP languages.
In this section we will discuss the zero knowledge proof for one such NP complete problem called as graph3 colouring problem.
A graph $G = (V, E)$ is said to be 3 colourable if there exist a $\phi : V \to \{0, 1, 2\}$ such that for every edge $(u, v) \in E,\ \phi(u) \neq \phi(v)$. now we will discuss the zero knowledge proof for 3-colouring problem.

Theorem 2.4.5. *If perfectly binding commitments exists then there exist a zero knowledge for 3 colouring problem.*

Proof. **start:** The prover P and the verifier V both of them will be having a graph G and P will be having an assignment φ at the beginning of the protocol. A counter called **rounds** is initialised by V .
At the beginning of the protocol ,set rounds = 0.

1. P randomly chooses a permutation $\pi : \{0,1,2\} \to \{0,1,2\}$ out of 6 possible permutations. computes the commitments as

$$\forall v \in V,\ c_v \leftarrow comm(\pi(\varphi(v), r_v))$$

   and sends it to V.

2. Now V samples an edge(u, v) randomly and sends it to P.

3. after getting the edge from the V, P sends $(\pi(\varphi(u), r_u, \pi(\varphi(v), r_v)$ to the V.

4. after getting $(\pi(\varphi(u), r_u, \pi(\varphi(v), r_v)$ from the P, V will check the following :

   - $\pi(\varphi(u)), \pi(\varphi(v)) \in \{0,1,2\}$
   - $\pi(\varphi(u) \neq \pi(\varphi(v)$
   - $c_u = comm(\pi(\varphi(u), r_u))$
   - $c_v = comm(\pi(\varphi(v), r_v))$

If any of these conditions are not satisfied, then V immediately outputs reject. Otherwise, it sets rounds = rounds+1. If rounds $\approx |G|^3$ then accept. Otherwise it goes back to step 1. □

Please refer the article "Interactive proofs and zero knowledge"[6] for full proof of the above protocol.

2.5 Blockchain and Smartcontracts

In today's world Blockchain is playing a key role in establishing trust between untrusted parties because of its immutability and Decentralization [6]. In our setting also Blockchain will be useful for establishing the trust between the two untrusted parties buyer(B) and the owner(O). Blockchain is playing a key role in bringing an important aspect "Fairness" in the protocol. As we know that it is impossible to solve a strong fair exchange without a trusted third party [7] so here in our setting Blockchain is acting as a trusted third party. Smart contracts are playing a key role in the automatic verification of keys and transfer of money.

Chapter 3

The problem statement

There are two parties in the protocol called the data Owner O, and buyer B. Buyer will have Search word w, Reward money M,Hash of the file(h)which he wanted to buy. where as the seller will has the Files $\{F_1,\ F_2\ \}$ along with unique key words $\{w_1, w_2\}$ associated with them respectively.

Let D_w be the set of words, D_f be the set files respectively.

Input to the Buyer B: Search word $w \in \{0,1\}^*$, Reward money M.

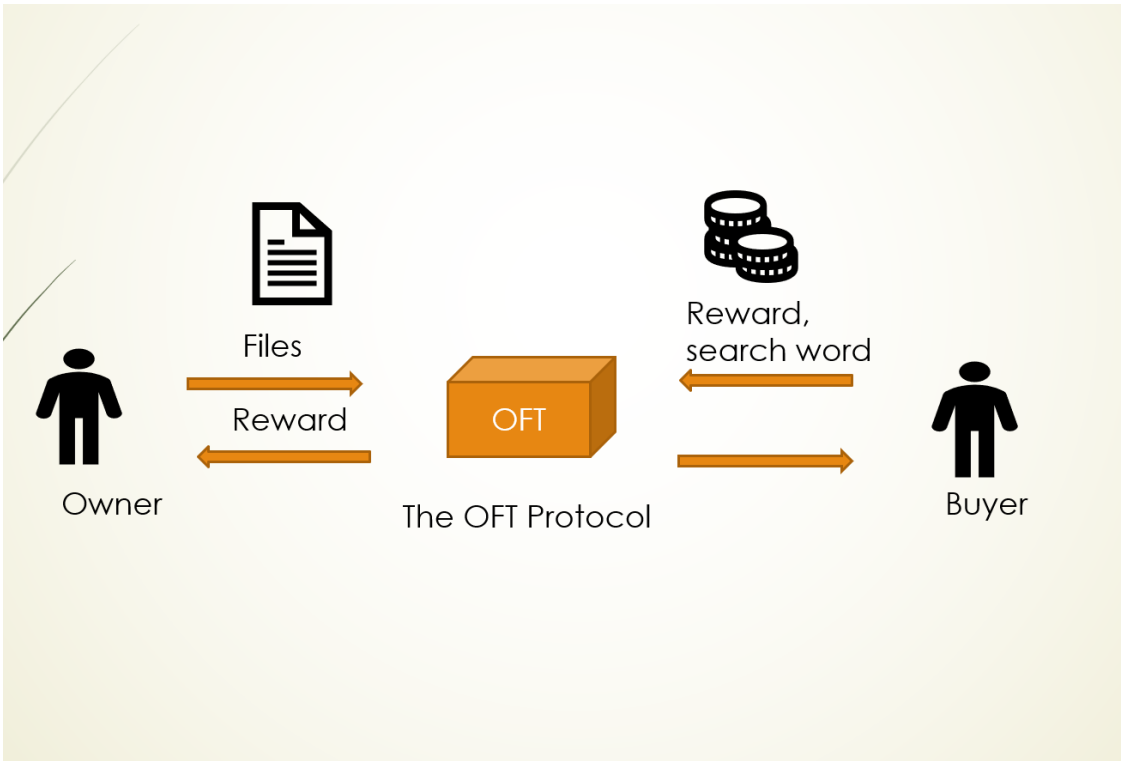
Input to the Owner O: The file set $\mathbf{F} = \{(F_i, w_i)|i \in [2]\}$. where $F_i \xleftarrow{\$} D_f$,$w_i \xleftarrow{\$} D_w$.

And a file $F_i = \{word_j|j \in [m_j]\}$ is assumed to be the set of distinct m_j words.

In this protocol buyer wants to buy a file whose hash value(h) he already knows, and he has a search word(w) for searching the file. Buyer will search for the file with out revealing the search word to the Owner.

There are 2 Possible input types for this protocol:

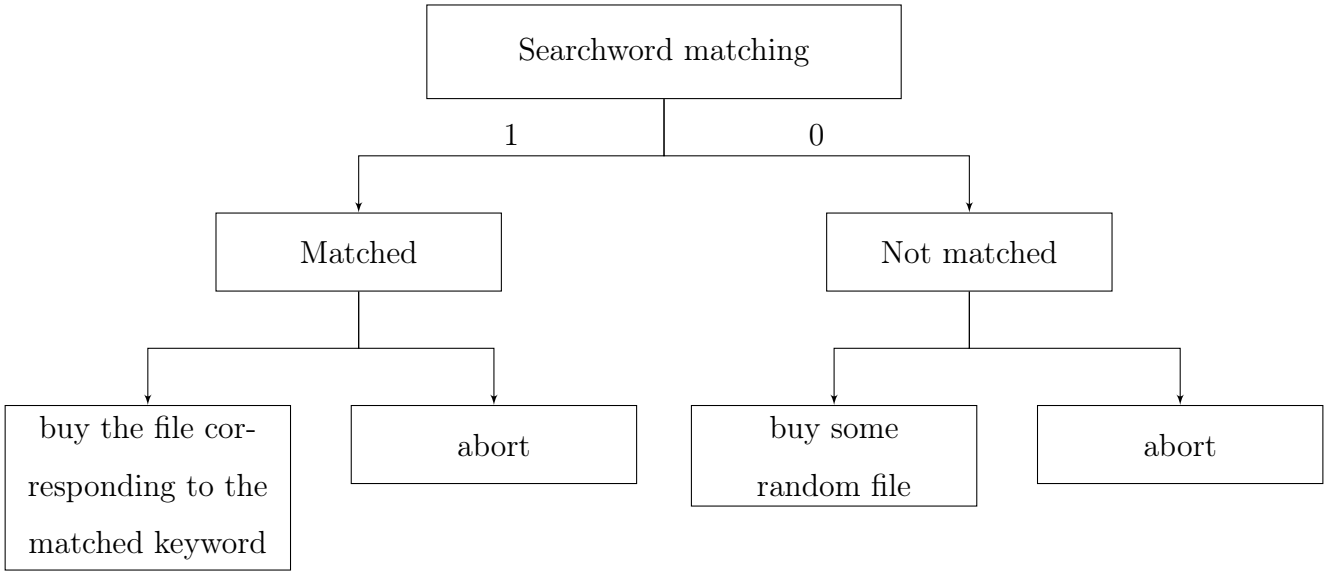
1. $w \in \{w_1, w_2\}$- Search word matches with the one of the keyword.



2. $w \notin \{w_1, w_2\}$- Search word does not match with any of the keyword.

The following are the expected outputs from the protocol depending on the type of input:

1.If the search word matches the key word then B will have two options either he can buy the file for which the search word matched with the key word or he can abort the protocol this is called decision step. B can choose one of the two options either buy or abort as his decision.



1. If the search word matches the with the key word, and if the buyer wants to buy the file which is corresponding to the matched key word then buyer should get the file and owner should get the file.

2. if the search word didn't matched with any of the key word, but if the buyer still wants to buy some file from the owner should get the reward money and buyer will get some random file.

There are two main goals to achieve in this problem.

1. Privacy of the search word & files.

2. Fairness in the trading protocol.

let me explain these two things with little more details. By privacy of the search word we mean that till the end of the protocol owner should not get to know the

search word of the buyer. similarly buyer also should not get the file till owner gives an access to the files. The trading of the digital file will be successful only if the both the parties exchanges their things that is the buyer need to give the money to the owner and owner need to give his files to the buyer. But who should first give their things to the other party? If owner gives the files buyer might simply get the files and escape with out giving the money, similarly if the buyer first gives his money owner might escape with out giving files.

Chapter 4

A solution to the problem

Any solution to this problem should consist of two components.

1.Oblivious searching

2.Fair trading

we will achieve Oblivious searching by designing a new cryptographic primitive, namely *Oblivious data trading*(ODT) denoted by π, that guarantees both the search word and file privacy.

Two basic ingredients for ODT primitive are .

1.Bilinear pairing

2.Hash functions.

One of the main goal of the ODT protocol is to guarantee the search word$w$ is never revealed. If the Buyer directly gives the search word w to the Owner, search word privacy will be lost. So instead of directly giving he will create a tokentk in such a way that, from the token no one can get to know the search word w.

we can achieve this by using discrete logarithm property as follows: Let G be a cyclic

group generated by g in which discrete log problem is difficult to break. we hide the search word w behind the token tk by raising the generator element g with the search word w, to avoid the dictionary attacks as well as to make it owner dependent the session key

$$s_2$$

and public key pk are included in the exponent. So in this way it is difficult to get the search word from the token there by we can achieve the privacy of the search word. **There had been a protocol developed by 3 people, my super visor Dr.Souradyuti paul(IIT Bhilai) and two of his students Dr.Ananya srivastava(IIT Gandhinagar), Mohammad Sumair(IIT Bhilai) to solve this problem, But that protocol was having some issue so in this thesis I have fixed those problem by coming up with a New OFT protocol with the help of my supervisor. so lets discuss the prior work in this chapter and New protocol in the next chapter.**

4.0.1 The ODT primitive

In order to solve the *Oblivious Data Trading* problem/protocol , we now define a new cryptographic primitive, namely Oblivious Data Trading ODT. This primitive is adapted from an existing primitive, namely OKSA, to include additional security properties such as *fairness* and *public verification* for the comparison between OKSA and ODT primitives.) From a high level, an oft protocol allows a data owner to trade a file that contains the *search word* queried by the user. Also, every file is associated with a unique *keyword*.

DEFINITION. Suppose $\lambda \in \mathbb{N}$ is the security parameter. The ODT scheme $\Pi = (\Pi.\mathsf{KeyGen}, \Pi.\mathsf{Session}, \Pi.\mathsf{Bind}, \Pi.\mathsf{Enc}, \Pi.\mathsf{TokenGen}, \Pi.\mathsf{ParamGen}, \Pi.\mathsf{ParamVer}, \Pi.\mathsf{Match}, \Pi.\mathsf{Dec}, \Pi.\mathsf{ProofGen})$ is a 10-tuple of algorithms over the setup algorithm $\Pi.\mathsf{Setup}$ that works as follows.

The Π is parameterized by a file-set $\boldsymbol{F} := \{(F_{(i)}, \omega_{(i)}) \,|\, i \in [2]\})$, and the *search word* $\omega \in \{0,1\}^*$.

- $\Pi.\mathsf{Setup}(1^\lambda) \rightarrow pp$: On input the security parameter 1^λ, it returns the public parameter pp.
- $\Pi.\mathsf{KeyGen}(pp) \rightarrow (pk, sk)$: On input pp, it returns the public/private key-pair (pk, sk).
- $\Pi.\mathsf{Session}(pp) \rightarrow s_1$: On input pp, it returns the session key s_1. (It is called by parties O and U.)
- $\Pi.\mathsf{Bind}(pp, sk, s_1, \boldsymbol{F}) \rightarrow v$: On input pp, sk, s_1 and $\boldsymbol{F}$, it returns v, which binds the owner's identity, session key, and file-set. (It is invoked by party O.)
- $\Pi.\mathsf{Enc}(pp, sk, s_1, \boldsymbol{F}) \rightarrow \boldsymbol{C}$: On input pp, sk, s_1, and $\boldsymbol{F}$, it returns the ciphertext $\boldsymbol{C}$ that includes associated data. (It is invoked by party O.)
- $\Pi.\mathsf{TokenGen}(pp, (pk, \omega)) \rightarrow (tk, s_2)$: On input pp, pk, and ω, it returns the *search token* tk corresponding to (pk, ω) and the session key s_2. This tk allows *only* the party having pk to generate parameters, useful for two most important operations in any OFT protocol to be described later, namely, membership testing of $\omega \in \{w_{(1)}, w_{(2)}\}$, and decryption of $C \in \boldsymbol{C}$ to get the correct $F \in \boldsymbol{F}$, without knowing ω. (It is invoked by party U.)

- $\Pi.\mathsf{ParamGen}(pp, sk, s_1, tk, \boldsymbol{F}) \rightarrow (param_1,\ param_2)$: On input pp, tk, sk, s_1, and $\boldsymbol{F}$, it returns a pair of parameters $(param_1, param_2)$, where: $param_1$ is used to test whether $\omega \in \{w_{(1)}, w_{(2)}\}$; and $param_2$ will be used for decrypting $C \in \boldsymbol{C}$ to get the correct $F \in \boldsymbol{F}$, without knowing ω. (It is invoked by party O.)

- $\Pi.\mathsf{ParamVer}(pp, v, tk, param_1) \rightarrow b$: On input pp, v, tk and $param_1$, it returns a bit b, if $b = 0$ then the 3-tuple $(v, tk, param_1)$ is not correctly generated. (It is invoked by party U and tk includes U's session key s_2.)

- $\Pi.\mathsf{Match}(pp, s_2, C, param_1) \rightarrow proof$: On input pp, s_2, C, and $param_1$, it returns $proof$, where $proof \neq \perp$ *iff* $\omega \in \{\omega_{(1)}, \omega_{(2)}\}$. Additionally, the $proof$ will be used later by the third party to decrypt C. The pair $(C, proof)$ will be used by BB to authenticate $proof$.(It is invoked by party U.)

- $\Pi.\mathsf{Dec}(pp, s_2, param_2, C) \rightarrow F'$: On input pp, s_2, $param_2$, and C, it returns the decrypted file F'. (It is invoked by party U.)

  Remark: C is decrypted in two different ways: (i) privately/locally using $param_2$ and s_2; (ii) publicly using $proof$ and s_1. Note that revealing s_2 reveals ω which is prohibited under any circumstances, therefore, in the dispute s_1 is revealed. If s_1 is revealed then F is revealed but it is less of a problem than revealing ω. It seems that we need a major mathematical discovery in order to be do the verification without revealing F.

- $\Pi.\mathsf{ProofGen}(pp, sk, s_1, proof, \boldsymbol{F}) \rightarrow param_3$: On input pp, sk, s_1, $proof$, and $\boldsymbol{F}$, it returns $param_3$. (It is invoked by party O.)

  Note that for a third party to settle any dispute, it needs to do the following operations: (i) $proof$ and s_1 together will be used for decrypting C to get F.

Note that the third party requires authentic information from both the parties in order to complete the decryption; and (ii) $param_3$ is used to verify $\omega \in F$. Now, the third party gets all the required data for verification from the state st of BB . If the verification fails at this point then the onus is on the owner because $proof$ is computed in authentic manner (it can be checked in the BB algorithm by matching it against the associated data contained in C computed by the owner himself).

4.1 The Protocol: OFT

In this section, we design a oblivious and fair trading protocol OFT – using the primitives ODT and BB – that guarantees *privacy* and *fairness* security in the malicious model with the dishonest majority. The OFT protocol – denoted by Γ – works in BB-hybrid model. The algorithmic (and pictorial) description of Γ is given in the section. 4.1.1.

We have already described the parties inputs and their respective outputs in Sect. 3. In addition, O and U have the current state $st = \{\mathsf{coins}_U(M+Pl), \mathsf{coins}_O(Pl)\}$. Here, $\mathsf{coins}_X(Y)$ denotes that, in the state st, the amount owned by party X is $Y \in N$; also $\mathsf{coins}(Y)$ denotes Y amount of cryptocurrencies independent of the owner.

4.1.1 Description of Protocol Γ

$\Gamma[pp, st, \Pi, \mathsf{BB}]$

Input: Data owner O has: file set $\boldsymbol{F} = \{(F_{(1)}, \omega_{(1)}), (F_{(2)}, \omega_{(2)})\}$. User U has: *search word* ω. Both the parties have the *initial* state $st = \{\mathsf{coins}_U(R + Pl), \mathsf{coins}_O(Pl)\}$.

Output If $\omega = \omega_{(j)}$ and $\omega \in F_{(j)}$ (for some $j \in [2]$) then: U receives $F_{(j)}$; O receives $\mathsf{coins}(R)$. If $\omega = \omega_{(j)}$ and $\omega \notin F_{(j)}$ (for some $j \in [2]$) then: U receives $F_{(j)}$ and $\mathsf{coins}(Pl)$; O receives $\perp$. Otherwise, both the parties receives $\perp$.

Stage 1: Setup

1. **[O generates keys (Offline)]** O invokes $\Pi.\mathsf{KeyGen}(pp) \rightarrow (pk_O, sk_O)$; and then invokes $\Pi.\mathsf{Session}(pp) \rightarrow s_O$.

2. **[U starts the protocol (Online)]** U invokes $\mathsf{BB}.\mathsf{Start}_U() \rightarrow (st, \beta_0)$.

Stage 2: Data Commitment

3. **[O binds session, owner and file set (Offline)]** O invokes $\Pi.\mathsf{Bind}(pp, sk_O, s_O, \boldsymbol{F}) \rightarrow v$.

4. **O generates ciphertexts and stores data (Online)]** O invokes $\Pi.\mathsf{Enc}(pp, sk_O, s_O, \boldsymbol{F}) \rightarrow \boldsymbol{C}$, where $\boldsymbol{C} = (C^{(1)}, C^{(2)})$. Then, O stores: $\mathsf{BB}.\mathsf{Store}\big((\boldsymbol{C}, v)\big) \rightarrow (st, \beta_1)$.

   [Here $C^{(j)}$ is the ciphertext for $(F_{(j)}, \omega_{(j)})$ which includes associated data such as authentication data, etc.]

5. **[U generates (and stores) search token (Online)]** U invokes $\Pi.\mathsf{TokenGen}(pp, (pk_O, \omega)) \to (tk, s_U)$. Then, U stores tk: $\mathsf{BB.Store}(tk) \to (st, \beta_2)$.

   [Here, tk denotes the search token which will be be used later to *obliviously* detect whether $\omega \in \{\omega_{(1)}, \omega_{(2)}\}$.]

6. **[O generates (and stores) *parameters* (Online)]** O invokes $\Pi.\mathsf{ParamGen}(pp, sk_O, s_O, tk, \boldsymbol{F}) \to (param_1, param_2)$. Then, O stores $param_1$ (only): $\mathsf{BB.Store}(param_1) \to (st, \beta_3)$.

   [Here, $param_1$ will be used to *obliviously* detect whether $\omega \in \{\omega_{(1)}, \omega_{(2)}\}$; and $param_2$ will be used for oblivious decryption. Note that $param_2$ is not stored in this step.]

Stage 3: Verification

7. **[U verifies $\boldsymbol{param_1}$ (Offline)]** U invokes $\Pi.\mathsf{ParamVer}(pp, v, tk, param_1) \to b \in \{0, 1\}$. If $b = 0$ then U *aborts*.

   [Note that $b = 1$ implies that $param_1$ correctly *corresponds* to $\boldsymbol{F}$, and ω (as well as O's secret key sk_O and the sessions s_O and s_U).]

8. **[U verifies whether $\boldsymbol{\omega \in \{\omega_{(1)}, \omega_{(2)}\}}$ (Offline)]** For all $j \in [2]$: U invokes $\Pi.\mathsf{Match}(pp, s_U, C^{(j)}, param_1) \to proof_j$; if $proof_j \neq \perp$ then $proof := proof_j[0]$, $C := C^{(j)}$.

   [Note that $proof_j \neq \perp$ *iff* $(\omega \in \{\omega_{(1)}, \omega_{(2)}\})$.]

Stage 4: Pledging of reward and penalty

9. **[U pledges reward and penalty (Online)]** If $proof_0 \neq \perp$ or $proof_1 \neq \perp$ then U invokes $\mathsf{BB.Pledge}_U() \rightarrow (st, \beta_4)$. Otherwise, U *aborts*. $\mathsf{BB.Pledge\_Store}_O(param_2) \rightarrow (st, \beta_5)$.

10. **[O pledges penalty and stores $param_2$ (Online)]** O invokes $\mathsf{BB.Pledge\_Store}_O(param_2) \rightarrow (st, \beta_5)$.

Stage 5: File Decryption

11. **[U recovers the file (Offline)]** If $\beta_5 = 0$ then go to step 14. Otherwise, U invokes $\Pi.\mathsf{Dec}(pp, s_U, param_2, C) \rightarrow F'$; parse $F' = F'\|\omega'$; if $(\omega' = \omega)$ then U *aborts*.

Stage 6: Public Verification

12. **[U claims incorrect file (Online)]** U sends $proof$: $\mathsf{BB.Challenge}_U(proof) \rightarrow (st, \beta_6)$.

    [Here, $proof$ ensures that $\omega \in \{\omega_{(1)}, \omega_{(2)}\}$, however, it does not ensure that F' does not contains search word, and it will be used to decrypt C publicly.]

13. **[O counters challenge (Online)]** If $\beta_6 = 0$ then go to step 14. Otherwise: O invokes $\Pi.\mathsf{ProofGen}(pp, sk_O, s_O, proof, \boldsymbol{F}) \rightarrow param_3$; and O stores $param_3$: $\mathsf{BB.Response}_O(s_O, param_3) \rightarrow (st, \beta_7)$.

    [Note that $param_3$ will be verified in $\mathsf{BB.Response}_O(\cdot)$.]

Stage 7: Redeem Reward

14. **[Party X redeem reward/refund (Online)]** X invokes $\mathsf{BB.Redeem}() \rightarrow (st, \beta_8)$.

From the standpoint of design, the Γ consists of 7 stages, all executed by parties O and U.

Stage 1: Setup. The purpose of this stage is to generate parameters for executing the protocol as well as starts the protocol execution time. The public parameter pp is used by O to generate the keys: private/public key pair (pk_O, sk_O); and session key s_O.

In this stage, U starts the protocol by invoking $\mathsf{BB.Start}_U(\cdot)$ that sets the starting time of the protocol using global clock. The description of stage 1 is given in steps 1 and 2 of section 4.1.1.

Stage 2: Data Commitment. The purpose of this stage is to commit file-set, search word and parameters for matching. In this stage, initially, before the verification of any of the conditions start, O does the following: (i) binds the session key and file set with his identity, which we call as v; then (ii) hides keywords, files, encryption key (symmetric) and session key, which we call as $\boldsymbol{C}$. Note that the encryption key is hidden using session key in order to prevent revelation of the encryption key as an output of the verification process in stage 3. Finally, O shares $(\boldsymbol{C}, v)$ with U through BB.

At this stage, U also hides his search word ω behind some quantity, which we call the token tk and stores it in BB. Note that in order to avoid the dictionary attack, tk has to be a function of a session key s_U. Also, in order to make the entire search process owner dependent, tk needs to be a function of pk_O as well. This allows U to

ban specific data owners from selling data to him.

Finally, O generates a pair of parameters, which we call as $(param_1, param_2)$, using token tk and stores $param_1$ in BB. Here, $param_1$ is required to obliviously validate if the *search word* matches the *keyword*, and $param_2$ is required to obliviously decrypt the matched file. Note that at the end of this stage U only knows the parameter for matching but not for decrypting the matched file.

Stage 3: Verification. The purpose of this stage is to *obliviously* match the *search word* with the *keyword* in such a way that O will not know about the *search word* if it does not matches the *keyword*. In this stage, U verifies if $param_1$ correctly corresponds to F and ω (as well as the keys sk_O, s_O, s_U) using tk and v. If the validation is successful then U *obliviously* checks if $\omega \in (\omega_{(1)}, \omega_{(2)})$ using $param_1$ and his session key s_U. In case of match, U will receive $proof$ that will be used later in BB to decrypt the matched file C, in case of dispute. The description of stage 3 is given in steps 7 and 8 of section. 4.1.1.

Stage 4: Pledging of reward and penalty. The purpose of this stage is to allow parties to pledge reward and penalty amount. Once U finds a match in stage 3, he pledges the reward to O using BB which can be redemeed after the timelock. Then, O stores $param_2$ in BB (along with the penalty amount which will be used later in case of dispute) so that U can decrypt the matched file. The description of stage 4 is given in steps 9 and 10 of section. 4.1.1.

Stage 5: File Decryption The purpose of this stage is to decrypt the file. In this stage, U decrypts the matched file using $param_2$. If the decrypted file contains the searchword then U stops executing the protocol and after timelock O executes stage 7 and gets the reward. If the decrypted file dos not contain the searchword then

U resolves the dispute by executing challenge-response protocol in Stage 6. It is important to note here that if the decrypted file contains the *searchword* then O will *never* learn the private input of the user. The description of stage 5 is given in step 11 of section 4.1.1.

Stage 6: Public Verification. The purpose of this stage is to penalize O if the file does not contain the *search word*. If the file decrypted by U does not contain the *search word*, then he challenges O to publicly prove the validity of the file. This is done by posting in BB the challenge value $proof$ which proves that U has certainly found a match. Now, O proves the validity of the file by posting in BB the response value $param_3$ and his session key s_O. If the following public validations are successful then O will be penalized: $(C, proof)$ correctly authenticates $proof$; and upon decryption (using $proof$ and s_1) the oblivious matching of the search word with the keyword in the file using $param_3$ returns true. In case, O fails to post response within the timelock then also he will be penalized. The description of stage 6 is given in steps 12 and 13 of Section. 4.1.1.

Stage 7: Redeem Reward The purpose of this stage is to allow a party to redeem the money after the timelock. If the protocol has executed correctly then after the timelock O can redeem the reward amount. In case, U receives incorrect file then he gets penalty amount as well as refund of reward amount. Also, if U wrongly challenges O then he pays penalty amount as well the reward amount to O. The description of stage 7 is given in step 14 in section. 4.1.1.

4.2 Building the ODT primitive

In this section we will give the construction details of the ODT.

- $\Pi.\mathsf{Setup}(1^\lambda) \rightarrow pp$: On input the security parameter 1^λ, where $\lambda \in \mathbb{N}$, it outputs the public parameter $pp = (p, \mathbb{G}, \mathbb{G}_T, g, g_T, e, H, \tau, M, Pl)$.

  [Here, p is a large prime; $(\mathbb{G}, \cdot)$ and $(\mathbb{G}_T, \times)$ denote groups of prime order p having generators g and g_T; $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ denotes a bilinear map; $H : \mathbb{G}_T \rightarrow \{0,1\}^l$ is a collision-resistant hash function; τ denotes the time parameter of the protocol; and $M, Pl \in \mathbb{N}$ denote the reward and penalty amounts.]

- $\Pi.\mathsf{KeyGen}(pp) \rightarrow (pk, sk)$: On input pp, it outputs (pk, sk) which is computed as follows: first, choose a random number α from $\mathbb{Z}_p$; and then initialize the secret key sk with α, and public key pk with g^α. The pseudocode is given in Fig. 4.1.

$\underline{\Pi.\mathsf{KeyGen}(pp)}$

\# Generates public-private key pair randomly
$\alpha \xleftarrow{\$} \mathbb{Z}_p$
$sk := \alpha$, and $pk := g^\alpha$
return (pk, sk)

Figure 4.1: Algorithmic Description of $\Pi.\mathsf{KeyGen}$

- $\Pi.\mathsf{Session}(pp) \rightarrow s_1$: On input pp, it outputs the session key s_1 randomly chosen from $\mathbb{Z}_p$. The pseudocode is given in Fig. 4.2.

Π.Session(pp)

\# Generates session key randomly
$s_1 \xleftarrow{\$} \mathbb{Z}_p$
return s_1

Figure 4.2: Algorithmic Description of Π.Session

- Π.Bind$(pp, sk, s_1, \boldsymbol{F}) \rightarrow v$: On input pp, sk, s_1 and $\boldsymbol{F}$, it outputs v which is computed as follows: first, bind all the keywords, $\omega_{(1)}$ and $\omega_{(2)}$, using owner's identity sk; then compute inverse of it $[(sk + \omega_{(1)})(sk + \omega_{(2)})]^{-1}$; finally, after combining the inverse value with session key, raise it with the group element to obtain v. The pseudocode is given in Fig. 4.3.

  [We assume that $\omega_{(1)}, \omega_{(2)} \in \mathbb{Z}_p$. We also assume there exists a public *injective* function $f(\cdot)$ that maps an actual keyword in $\mathcal{KS}$ to $\mathbb{Z}_p$, where $\mathcal{KS} \in \{0,1\}^l$ and $l \leq \lfloor \log(p-1) \rfloor$.]

Π.Bind$(pp, sk, s_1, \boldsymbol{F} = \{(F_{(1)}, \omega_{(1)}), (F_{(2)}, \omega_{(2)})\})$

\#Binds the owner's identity, session key, and file-set.
$v := g^{s_1[(sk+\omega_{(1)})(sk+\omega_{(2)})]^{-1}}$
return v

Figure 4.3: Algorithmic Description of Π.Bind

- Π.Enc$(pp, sk, s_1, \boldsymbol{F}) \rightarrow \boldsymbol{C}$: On input pp, sk, s_1, and $\boldsymbol{F}$, it outputs the ciphertext $\boldsymbol{C}$ along with its associated data which is computed as follows: For each file in file-set, first, choose the encryption key randomly, denoted $r^{(i)}$; then compute the associated data $h^{(i)}$ which is a tuple of two values, the first value hides the keyword $\omega_{(i)}$ by raising the group element with $(r^{(i)}, s_1, sk, \omega_{(3-i)})$ and the

second value commits $(r^{(i)}, s)$; then encrypts file $F_{(i)}\|\mathsf{bin}(\omega_{(i)})$ using encryption key $r^{(i)}$ to generate ciphertext $\widetilde{C}^{(i)}$; finally it initializes $C^{(i)}$ with $(\widetilde{C}^{(i)}, h^{(i)})$. Here the function $\mathsf{bin}(\cdot)$ takes an integer as input and converts it into binary. The pseudocode is given in Fig. 4.4.

$\Pi.\mathsf{Enc}(pp, sk, s_1, \boldsymbol{F} = \{(F_{(1)}, \omega_{(1)}), (F_{(2)}, \omega_{(2)})\})$

Encrypts file-set

- **for** $i \in [2]$ **do** $r^{(i)} \xleftarrow{\$} \mathbb{Z}_p$

$h^{(i)} := \left(g^{r^{(i)} s_1 (sk + \omega_{(3-i)})},\ H(e(g,g)^{r^{(i)} s_1^2}) \right)$

$\widetilde{C}^{(i)} := \left(H(e(g,g)^{r^{(i)}}) \oplus \left(F_{(i)}\|\mathsf{bin}(\omega_{(i)})\right) \right)$

$C^{(i)} := (\widetilde{C}^{(i)}, h^{(i)})$

$\boldsymbol{C} := (C^{(1)}, C^{(2)})$

return $\boldsymbol{C}$

Figure 4.4: Algorithmic Description of $\Pi.\mathsf{Enc}$

- $\Pi.\mathsf{TokenGen}(pp, (pk, \omega)) \rightarrow (tk, s_2)$: On input pp, pk, and ω, it outputs token tk which is computed as follows: first choose the session key s_2 randomly from $\mathbb{Z}_p$; then it initializes tk with $(pk \cdot g^{\omega})^{s_2}$ that hides the search word ω for a specific owner having pk. The pseudocode is given in Fig. 4.5.

$\Pi.\mathsf{TokenGen}(pp, (pk = g^{\alpha}, \omega))$

Hides search word

$s_2 \xleftarrow{\$} \mathbb{Z}_p$

$tk := (pk \cdot g^{\omega})^{s_2}$

return (tk, s_2)

Figure 4.5: Algorithmic Description of $\Pi.\mathsf{TokenGen}$

- Π.ParamGen$(pp, sk, s_1, tk, \boldsymbol{F}) \rightarrow (param_1,\ param_2)$: On input pp, tk, sk, s_1, and $\boldsymbol{F}$, it outputs a pair of parameters $(param_1, param_2)$ which is computed as follows: first compute $param_1$ from the token tk by raising tk with s_1 and $(sk + \omega_{(1)})(sk + \omega_{(2)})]^{-1}$ (Note that if $\omega \in \{\omega_{(1)}, \omega_{(2)}\}$ then ω will get cancelled in this process); then compute $param_2$ from $param_1$ by raising it with the inverse of s_1 which nullifies the existence of s_1 from $param_1$ and returns the decryption key. The pseudocode is given in Fig. 4.6.

Π.ParamGen$(pp, sk, s_1, tk, \boldsymbol{F} = \{(F_{(1)}, \omega_{(1)}), (F_{(2)}, \omega_{(2)})\})$

\# Generates parameters for matching and decryption
$param_1 := tk^{s_1[(sk+\omega_{(1)})(sk+\omega_{(2)})]^{-1}}$
$param_2 := param_1^{(s_1^2)^{-1}}$
return $(param_1, param_2)$

Figure 4.6: Algorithmic Description of Π.ParamGen

- Π.ParamVer$(pp, v, tk, param_1) \rightarrow b$: On input pp, v, tk and $param_1$, it outputs a bit b. If the bilinear pairing between v and tk maps to the same element as the bilinear pairing between g and $param_1$ then the algorithm returns $b = 1$. Thus, by exploiting the *bilinear* property of the map, it is verified if $param_1$ is generated correctly using tk and the committed file-set. The pseudocode is given in Fig. 4.7.

Π.ParamVer$(pp, v, tk, param_1)$

\# Verifies the tuple $(v, tk, param_1)$
$e(v, tk) = e(g, param_1)$ return $b = 1$ return $b = 0$

Figure 4.7: Algorithmic Description of Π.ParamVer

- $\Pi.\mathsf{Match}(pp, s_2, C, param_1) \rightarrow proof$: On input pp, s_2, $C = (c_0, c_1)$, and $param_1$, it outputs $proof$ which is computed as follows: first, compute the pairing between $param_1$ and $c_1[0]$ (where $c_1[0]$ corresponds to the first component of h); then verify the commitment with $c_1[1]$ (where $c_1[1]$ corresponds to the second component of h); finally if verification is successful then $proof$ is initialized with a tuple of two values, first value is the pairing between $param_1$ and $c_1[0]$ and second value is the matched ciphertext c_0. The pseudocode is given in Fig. 4.8.

  Suppose $C = C^{(1)}$ and $\omega = \omega_{(1)}$ then pairing between $param_1$ and $c_1[0]$ nullifies the presence of all the keywords and returns the pairing between $(r^{(1)}, s)$ which is the pre-image of $c_1[1]$.

$\underline{\Pi.\mathsf{Match}(pp, s_2, C = (c_0, c_1), param_1)}$

\# Performs matching of search word with keyword
$c_1[1] = H\big(e(c_1[0], param_1)^{s_2^{-1}}\big)$
$proof := \big(e(c_1[0], param_1)^{s_2^{-1}}, c_0\big)$
return $proof$ return $\perp$

Figure 4.8: Algorithmic Description of $\Pi.\mathsf{Match}$

- $\Pi.\mathsf{Dec}(pp, s_2, param_2, C) \rightarrow F'$: On input pp, s_2, $param_2$, it outputs the decrypted file F'. The decryption is done as follows: first, compute the pairing between $param_2$ and $c_1[0]$ (which corresponds to the first component of h); compute its commitment; finally XORed it with the matched ciphertext c_0 to get (F', ω'). Note that $param_2$ removes s_1 from $param_1$, thus, the bilinear pairing between $param_2$ and $c_1[0]$ will now return the resultant key for decrypting *only* the matched file. The pseudocode is given in Fig. 4.9.

- $\Pi.\mathsf{ProofGen}(pp, sk, s_1, proof, \boldsymbol{F}) \rightarrow param_3$: On input pp, sk, s_1, $proof$ =

$\Pi.\mathsf{Dec}(pp, s_2, param_2, C = (c_0, c_1))$

\# Decrypting matched file

$F' := H\Big(e\big(c_1[0], param_2\big)^{(s_2)^{-1}}\Big) \oplus c_0$

return F'

Figure 4.9: Algorithmic Description of $\Pi.\mathsf{Dec}$

(p', c'), and $\boldsymbol{F}$, it outputs $param_3$ which is computed as follows: first it decrypts c' using p' and s_1; then it parses keyword ω'' from the decrypted file; since ω'' is in binary so the function $\mathsf{int}(\cdot)$ converts it into integer; finally $param_3$ is computed using similar technique like $param_1$ except that it is computed using ω'' instead of tk. The pseudocode is given in Fig. 4.10.

$\Pi.\mathsf{ProofGen}(pp, sk, s_1, proof = (p', c'), \boldsymbol{F} = \{(F_{(1)}, \omega_{(1)}), (F_{(2)}, \omega_{(2)})\})$

\# Computing response parameter

$F' := H\big(p'^{(s^2)^{-1}}\big) \oplus c'$

Parse $F' = F''\|\omega''$

$param_3 := g^{(sk+y)\, s_1[(sk+\omega_{(1)})(sk+\omega_{(2)})]^{-1}}$, where $y = \mathsf{int}(\omega'')$

return $param_3$

Figure 4.10: Algorithmic Description of $\Pi.\mathsf{ProofGen}$

4.3 Issue with the first developed OFT protocol

The issue with the first developed protocol.

1. If the buyer is dishonest, Even though he gets the correct file(the file which contains the search word) he will falsely claims that he got a wrong file, so he

will force the owner to go to the public verification phase , But in the public verification phase there is no way to prove that the encrypted file contains the search word with out revealing the search word, and without decrypting the file. So there is no way other than decrypting the file. There are two ways to decrypt the files.

(a) Using $param2$ and s_2(user's session key)

(b) using $proof$ and s_1.

If s_2 is revealed then any one can run the matching algorithm and get to know the search word. which is breaking the privacy of the search word.
If s_1 is revealed then any one can run the decryption algorithm and get to know the files.Which is breaking the privacy of file.

Chapter 5

The New Solution to the Problem

5.1 Structure of the new protocol

There are 4 stages.

Stage 1: Key generation We generate all the required keys.

Abort condition: Time out

Stage 2: Oblivious searching/matching Buyer is searching for his search word through the keywords of the owner. That has to be done obliviously. If matching fails then the buyer can abort.

Abort condition: Time out+ verification algorithm outputs 0

Stage 3A: Owner transfers the files obliviously & **Buyer verifies the correctness of the files**

The verification ensures that the O transferred the correct file in the hidden form or not.

Abort condition: Time out+verification algorithm outputs 0.

Stage 3B: Timelocked commitment of money .

Abort condition: Time out

Stage 4:storage & **verification of decryption keys and transferring of money**

Storage of decryption keys by the owner and their automatic verification. If verified then the committed money is transferred, otherwise refunded.

Abort condition: Time out

.

5.2 The New Protocol:OFT

In this section, we design a new oblivious and fair trading protocol OFT in which both the privacy and fairness are guaranteed.

Description of the Protocol Γ

Input: Data owner O has: file set $\mathbf{F} = \{(F_1, w_1), (F_2, w_2)\}$.

Buyer B has: search word w, money M, hash of the file which he wanted to buy h.

Output: If $w_i = w$ (for some $i \in [2]$) then B receives F_i; O receives the money M.

If $w_i \neq w$ for any $i \in [2]$ then B can abort the protocol.If B aborts the protocol then exchange of the file and the money doesn't take place.

If B continues with the protocol with out aborting it then B gets some file which might not be the one which he wanted to buy; O will get the money M.

Stage 1:Key generation

[O **generates keys**] O invokes KeyGen $(pp) \rightarrow (pk, sk)$.

Stage 2:Oblivious searching.

This stage has the following steps.

Step 1:Commitment

[O **commits keywords**] O invokes commit$(w_1, w_2) = v$

Step 2:O sends commitments to B

O $\xrightarrow{v}$ B

Step 3:O creates a token

O invokes $T_1(w_1, w_2, S_1, sk) \rightarrow tk_O$. where s_1 is the owner's session key.

O sends this token to B. $O \xrightarrow{tk_O} B$

Step 4:

[B **creates the token from the search word**] **B invokes** $T_2(w, pk, s_2) \rightarrow tk_B$.

where s_2 is the buyer's session key

B sends this token to O. $B \xrightarrow{tk_B} O$

Step 5:O creates a parameter and sends it to B.

O invokes $T_3(w_1, w_2, S_1, sk, tk_B) \rightarrow param$.

O sends this parameter to B. $O \xrightarrow{param} B$

Step 6: B verifies the parameter.

B invokes verification$(Param, com_1, com_2, tk_o, tk_B) \rightarrow b \in \{0, 1\}$

b=1 means correct information went into param.

Step 7:

B invokes Matching$(tk_B, param, tk_O, com_1, com_2) \rightarrow (1, index\ i)$ where $i \in \{1, 2\}$ *or* 0.

If search word matches with the keyword it outputs the index of the matched key word.

If the search word doesn't match the keyword it outputs $(0, \perp)$.

Stage 3:

oblivious transfer of the files F_1, F_2 & verification of correctness of the files

. There are two components in this stage.

A Owner Obliviously transfers the file

A.1 **Encryption:** O encrypts the files F_i with key k_i for $i \in [1, 2]$

$$Enc_{k_i}(F_i) = C_i \ for \ i \in [1, 2]$$

.

A.2 **Commitment:** O commits the keys and the files.

$$Comm(k_i) = com_{k_i} \ for \ i \in [1, 2]$$

$$Comm(F_i) = com_{F_i} \ for \ i \in [1, 2]$$

.

A.3 **Onchain communication:** O stores $com_{k_1}, \ com_{k_2}$ on the Blockchain.

A.4 **O computes ZKPs:**

$$proof_1 = ZKP(C_1, com_{k_1}, com_{F_1})$$

$$proof_2 = ZKP(C_2, com_{k_2}, com_{F_2})$$

A.5 **Offchain oblivious transfer: Input:** O has

$$T[1] = (C_1, com_{k_1}, com_{F_1}, proof_1)$$

$$T[2] = (C_2, com_{k_2}, com_{F_2}, proof_2)$$

B has $i \in [1, 2]$

Output:$T[i]$

.

A.6 **Verification:** B verifies the ZKP. verify$(C_i, com_{k_i}, com_{F_i}, proof_i) = b$ where $b \in \{0, 1\}$.

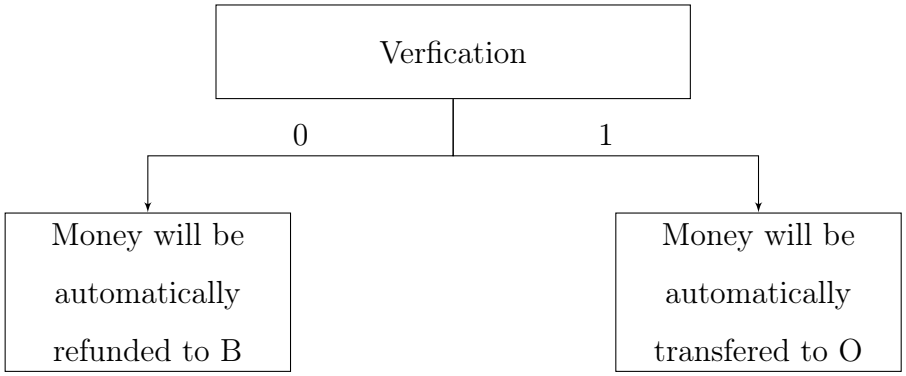
B If b=1 Buyer commits his money.

If b=0 Buyer aborts the protocol.

Stage4:

Storing,verification of Decryption keys & **transfer of money**

1. O stores both the decryption keys k_1 and k_2 on the Blockchain.
2. Verifcation of the decryption keys [Automatic].
3. Verification$(k_1, k_2) \rightarrow 0/1$.
4. If $comm(k_1) = com_{k_1}$ and $comm(k_2) = com_{k_2}$ verification algorithm outputs 1.



5.3 Algorithmic description of the New OFT Protocol

$\Gamma[pp, st, \mathsf{BB}]$

Input: Data owner O has: file set $\mathbf{F} = \{(F_1, w_1), (F_2, w_2)\}$.

Buyer B has: search word w, money M, hash of the file which he wanted to buy h.

Output: If $w_i = w$ (for some $i \in [2]$) then B receives F_i; O receives the money M.

If $w_i \neq w$ for any $i \in [2]$ then B can abort the protocol.If B aborts the protocol then exchange of the file and the money doesn't take place.

Stage 1: Setup

(a) **[O generates keys (locally)]** O invokes $\Gamma.\mathsf{KeyGen}(pp) \rightarrow (pk_O, sk_O)$; and then invokes $\Gamma.\mathsf{Session}(pp) \rightarrow s_1$.

O announces the public key to all.

(b) **[B sends the message "start the protocol" to the O](Offchain)**

Stage 2: Oblivious searching

(a) **[O binds session key, secrete key,keywords & sends it to B](offchain)** O invokes $\Gamma.\mathsf{Bind}(pp, sk_O, s_1, w_1, w_2) \to v$.
O sends v to the B.

(b) **[O generates tokens for keywords and sends it to B](Offchain)** O invokes $\Gamma.\mathsf{TokenGen1}(pp, sk_O, s_1, w_1, w_2) \to \boldsymbol{C}$, where $\boldsymbol{C} = (C_{(1)}, C_{(2)})$. and send it to B. [Here $C_{(j)}$ is the token for $w_{(j}$).

(c) **[B generates and send search token to O](offchain)** B invokes $\Gamma.\mathsf{TokenGen2}(pp, (pk_O, w)) \to (tk, s_2)$. Then, B sends tk to the O.
[Here, tk denotes the search token which will be be used later to *obliviously* detect whether $w \in \{w_{(1)}, w_{(2)}\}$.]

(d) **[O generates and sends *parameter* to the B](offchain)** O invokes $\Gamma.\mathsf{ParamGen}(pp, sk_O, s_1, tk, w_1, w_2) \to (param)$. Then, O sends $param$ to the B. [Here, $param$ will be used to *obliviously* detect whether $w \in \{w_{(1)}, w_{(2)}\}$.

(e) **[B verifies *param* (locally)]** B invokes $\Gamma.\mathsf{ParamVer}(pp, v, tk, param) \to b \in \{0, 1\}$. If $b = 0$ then B *aborts.*

[Note that $b = 1$ implies that *param* correctly *corresponds* to w_1, w_2, and w (as well as O's secret key sk_O and the sessions s_1 and s_2).]

(f) **[B verifies whether $w \in \{w_{(1)}, w_{(2)}\}$ (locally)]** For all $j \in [2]$: B invokes $\Gamma.\mathsf{Match}(pp, s_2, C_{(j)}, param) \rightarrow (0, \perp)$ *or* $(1, i)$.

[Note that *in tuple* $(1, i)$ 1 indicates$(w \in \{w_1, w_2\})$.]where i gives the index of the matched key word.

$(0, \perp)$ indicates that no key word got matched with the search word$w$.

Stage 3:

oblivious transfer of the files (F_1, F_2) & **verification of correctness of the files**

.

3A **Owner Obliviously transfers the file**

1. **[O encrypts and finds the hash of the keys** & **files (locally)]** For all $i \in [2]$ O invokes $\Gamma.Enc_{k_i}(F_i) \rightarrow C_i$ and computes

$$Enc(k_i, F_i) \rightarrow C_i$$

where k_i are secret keys.

along with the following Hash values:

$$H(k_i) = h_{k_i}$$

$$H(F_i) = h_i$$

2. **[O commits keys to the smart contract (Onchain)]** O sends $\{H(k_1), H(k_2)\}$ to the smart contract. which is acting as a trusted third party. O stores: $\mathsf{BB.Store}\big(H(k_1), H(k_2)\big) \rightarrow (st, \beta_1)$.

3. **[O computes ZKPs (locally)]** for $i \in [2]$ O invokes $\Gamma.ZKcompute(C_i, H(k_i), H(F_i)) \rightarrow proof_i$ for computing

$$proof_1 = ZKP(C_1, h_{k_1}, h_1)$$

$$proof_2 = ZKP(C_2, h_{k_2}, h_2)$$

4. **[Otransfers the ciphers obliviously](Offchain)** O invokes $\Gamma.OT(T[1], T[2]) \rightarrow T[i]$ where i is chosen by buyer$B$(index of the matched keyword) and

$$T[1] = (C_1, H(k_1), H(F_1), proof_1)$$

$$T[2] = (C_2, H(k_2), H(F_2), proof_2)$$

.

5. **[B verifies the ZKPs](locally)** B invokes $\Gamma.ZKVer(C_i, h_{k_i}, h_i, proof_i) \rightarrow b$ where $b \in \{0, 1\}$.

3.B **[B's Timelock commitment of money](onchain)** If b=1 in the previous step then B invokes $\mathsf{BB.Pledge}_B(t, M) \rightarrow (st, \beta_2)$. Otherwise, B *aborts.*

Stage 4

Storing,verification of Decryption keys & **transfer of money**

(a) **[O sends the decryption keys to the smart contract](on chain)** O invokes $\mathsf{BB.Pledge\_Store}_O(k_1, k_2) \rightarrow (st, \beta_3)$.

(b) **[Smart contract verifies the correctness of the keys](Automatic)** smart contract invokes $\Gamma.keyver(k_1, k_2, H(K1), H(k_2)) \rightarrow b$

(c) **[Smart contract automatically transfers the money](Automatic)** If $b = 1$ smart contract transfers the money to the O.
If $b = 0$ smart contract refunds the money to the B.

5.4 Description of the new protocol

In this section we will give the description of all the algorithms which we have used in the new protocol.

Stage 1**: Setup** $\Gamma.\mathsf{KeyGen}(pp) \rightarrow (pk_O, sk_O)$ this takes pp as an input and generates (pk_O, sk_O) as an output.

And Owner generates his session key s_1 by running $\Gamma.\mathsf{Session}(pp) \rightarrow s_1$.

After owner generating and sharing the public key with buyer ,Buyer sends the message "starts the protocol" if he wants to run the protocol with that owner

Stage 2**: Oblivious searching**

In this stage O generates tokens out of key words and B generates token out of search word and they both exchange their tokens each other.Now we can not match with

only these tokens there should be a common reference string, using which we can match the search word with the key word. so they will generate a parameter which acts as a common reference string which will be useful for matching search word with the keyword.

1. $\Gamma.\mathsf{Bind}(pp, sk_O, s_1, w_1, w_2) \rightarrow v$ using this O commits his keywords, secret keys, session keys

2. $\Gamma.\mathsf{Token\ gen1}(pp, sk_O, s_1,) \rightarrow \boldsymbol{C}$, where $\boldsymbol{C} = (C_{(1)}, C_{(2)})$. O generates the tokens out of his key words (w_1, w_2) which will be useful for oblivious matching of the keywords with the search word. This $\Gamma.\mathsf{Token\ gen1}$ alogorithm is same as $\pi.Enc$ alogoritm from the first developed protocol but only difference difference is here we are not encrypting the files we are just creating the token out of the search words.

3. using the $\Gamma.\mathsf{TokenGen2}(pp, (pk_O, w)) \rightarrow (tk, s_2)$ algorithm B creates a token out of his search word and sends this token to the O.This algorithm is exactly same as the $\pi.TokenGen$ algorithm from the first developed protocol.

4. In this step O generates a parameter using the $\Gamma.\mathsf{ParamGen}(pp, sk_O, s_1, tk, w_1, w_2) \rightarrow (param)$. This algorithm takes secrete key,session key of the owner, buyer's token, key words w_1, w_2 and generates a parameter.This algorithm is also same as the $\pi.paramGen$ from the first developed protocol but only difference is here we are generating only one parameter not two unlike the old protocol.

5. In this step B verifies the parameter generated by the O.
   $\Gamma.\mathsf{ParamVer}(pp, v, tk, param) \rightarrow b \in \{0, 1\}$.This algorithm is exactly same as

the $\pi.ParamVer$ algorithm from the first developed protocol. 4.2 The output $b = 1$means that the param is computed with the correct inputs.

6. In this step B runs the matching algorithm to check whether $w \in \{w_1, w_2\}$? This algorithm is slightly different than the $\pi.Matching()$ algorithm . This algorithm produces 1 or 0 corresponding to match or unmatch of the search word. If matches then it will outputs the index of the matched keyword$i$ otherwise $\perp$.

Stage 3:

oblivious transfer of the files F_1, F_2 & verification of correctness of the files

3A.1 O randomly chooses two keys k_1, k_2 and computes the following :

$Enc_{k_1}(F_1) = C_1$	$Enc_{k_2}(F_2) = C_2$
$H(k_1) = h_{k_1}$	$H(k_2) = h_{k_2}$
$H(F_1) = h_1$	$H(F_2) = h_2$

3A.2 O sends the h_{k_1}, h_{k_2} to the B.

3A.3 O computes the $ZKPs = (proof_1, proof_2)$ where $proof_i$ is the ZKP for the statements that

"C_i**is the Encryption of the** $File_i$ **encrypted with the key** k_i**",**

"h_{k_i} **is the Hash the key** k_i**"**

"h_i **is the Hash the File** F_i**"**

But how can we guarantee the existence of the above ZKPs?

For a given (F_i, k_i) we can verify the above statements in the polynomial time so this language is in NP. And we have proved in 2.4 that $\forall L \in$NP there exists

a Zeroknowledge proof.

Now owner has the following:

$T[1]$
$Enc_{k_1}(F_1) = C_1$
$H(k_1) = h_{k_1}$
$H(F_1) = h_1$
$proof_1 = ZKP(C_1, h_{k_1}, h_1)$

$T[2]$
$Enc_{k_2}(F_2) = C_2$
$H(k_2) = h_{k_2}$
$H(F_2) = h_2$
$proof_2 = ZKP(C_2, h_{k_2}, h_2)$

3A.4 Now O runs the $\Gamma.OT(T[1], T[2]) \rightarrow T[i]$ and sends exactly one $T[i]$out of them to the B.For more details about 1-2 oblivious transfer protocol refer 2.3

3A.5 B receives exactly one of $T[1], T[2]$ and verifies the ZKPs by running $\Gamma.ZKVer(C_i, h_{ki}, h_i, proof_i) \rightarrow b$ where $b \in \{0, 1\}$.

3B If $b = 1$ then B commits his money for time t, otherwise aborts.

Stage 4 **Storing,verification of Decryption keys** & **transfer of money**

In this stage O sends the decryption keys to the smart contract and smart contract checks correctness the keys with the help of the commitments. If it verifies it automatically transfers the money to the O, otherwise refunds the money to the B.

5.5 The construction details of the new OFT protocol

In this section we will give the construction details of the all the algorithms which we used in the new OFT protocol.

Some of the algorithms are exactly same as the algorithms in the first developed protocol which are discussed in the secton 4.2.

Here in this section we are only giving the details of the algorithms which are modified, or newly constructed.

stage1

The algorithms $\Gamma.KeyGen(), \Gamma.Session()$ algorithms are exactly same as the algorithms in the first developed protocol. for more details refer 4.2.

stage2

1. The algorithm $\Gamma.Bind(), \Gamma.TokenGen2(), \Gamma.ParamVer()$ are also same as in the old protocol.

2. The details of the $\Gamma.\mathsf{TokenGen1}(pp, sk_O, s_1,) \rightarrow \boldsymbol{C}$ are given below: This algorithm is a little modification of $\pi.Enc()$ algorithm

   $\Gamma.\mathsf{TokenGen1}(pp, sk, s_1, \omega_1, \omega_1) \rightarrow \boldsymbol{C_1, C_2}$: On input pp, sk, s_1, and ω_1, ω_1, it outputs the ciphertext $\boldsymbol{C_1, C_2}$ along with its associated data which is computed as follows: For each search word, first choose the encryption key randomly, denoted $r^{(i)}$; then compute the associated data $C_{(i)}$ which is a tuple of two values, the first value hides the keyword $\omega_{(i)}$ by raising the group element with $(r^{(i)}, s_1, sk, \omega_{(3-i)})$ and the second value commits $(r^{(i)}, s_1)$; The pseudocode is given in Fig. 5.1.

4. $\Gamma.\mathsf{ParamGen}(pp, sk_O, s_1, tk, w_1, w_2) \rightarrow (param)$ is a little modification of the $\pi.ParamGen()$ from the old protocol.

   $\Pi.\mathsf{ParamGen}(pp, sk, s_1, tk, w_1, w_2) \rightarrow (param)$: On input pp, tk, sk, s_1, and w_1, w_2, it outputs the parameter $(param)$ which is computed as follows: first

$\Gamma.\mathsf{TokenGen1}(pp, sk, s_1, \{\omega_1, \omega_2\})$

\# Generates the tokens for the keywords

3. **for** $i \in [2]$ **do** $r^{(i)} \xleftarrow{\$} \mathbb{Z}_p$

$C_{(i)} := \left(g^{r^{(i)} s_1 (sk + \omega_{(3-i)})},\ H(e(g,g)^{r^{(i)} s_1^2}) \right)$

$\boldsymbol{C} := (C_1, C_2)$

return $\boldsymbol{C}$

Figure 5.1: Algorithmic Description of $\Gamma.\mathsf{TokenGen1}$

compute $param$ from the token tk by raising tk with s_1 and $(sk + \omega_{(1)})(sk + \omega_{(2)})]^{-1}$ (Note that if $\omega \in \{\omega_{(1)}, \omega_{(2)}\}$ then ω will get cancelled in this process). The pseudocode is given in Fig. 5.2.

$\Gamma.\mathsf{ParamGen}(pp, sk, s_1, tk, \omega_{(1)}, \omega_{(2)}))$

\# Generates parameters for matching
$param := tk^{s_1[(sk+\omega_{(1)})(sk+\omega_{(2)})]^{-1}}$

return $(param)$

Figure 5.2: Algorithmic Description of $\Gamma.\mathsf{ParamGen}$

5. The algorithm $\Gamma.\mathsf{Match}(pp, s_2, C_{(j)}, param) \rightarrow (0, \perp)$ *or* $(1, i)$. is a little modification of the $\pi.Match()$ from the old protocol. $\Gamma.\mathsf{Match}(pp, s_2, C, param) \rightarrow (0, \perp)$ *or* $(1, i)$: On input pp, s_2, $C = (C_1, C_2)$, and $param$, it outputs $(0, \perp)$ *or* $(1, i)$ which is computed as follows: first, compute the pairing between $param$ and $c_i[0]$ (where $c_i[0]$ corresponds to the first component of the token C_i); then verify the commitment with $c_i[1]$ (where $c_i[1]$ corresponds to the sec-

ond component of the token); finally if verification is successful then it will give $(1, i)$ as an output . The pseudocode is given in Fig. 5.3.

$\Gamma.\mathsf{Match}(pp, s_2, C = (C_0, C_1), param)$

\# Performs matching of search word with keyword
If $C_i[1] = H\big(e(c_i[0], param)^{s_2^{-1}}\big)$ then return $(1, i)$
else return $(0, \perp)$

Figure 5.3: Algorithmic Description of $\Gamma.\mathsf{Match}$

Stage3:Algorithms

1. $\Gamma.Enc_{k_i}(F_i) \rightarrow C_i$ this can be any symmetric key encryption algorithm.
2. $\Gamma.OT(T[1], T[2]) \rightarrow T[i]$ is an 1-2 oblivious transfer protocol for more details refer 2.3
3. we don't know the exact $\Gamma.ZKcompute(C_i, H(k_i), H(F_i)) \rightarrow proof_i$
   and $\Gamma.ZKver(C_i, h_{k_i}, h_i, proof_i) \rightarrow b$ where $b \in \{0, 1\}$ algorithms but constructing such algorithms are possible.

Chapter 6

Conclusions and Future work

Conclusions:

Right now most of the trading protocols are not offering any kind of privacy and decentralization for example if we buy a movie from Google, the Google servers will get to know our search word and collect all our private data and uses this data for many purposes. So currently there is an extensive need for work on protocols which guarantees the privacy of the data. Similarly, If we buy a song with the most of the existing protocols first, we need to pay the money, we will get the song only after the payment is successful some times it might happen that after paying we may realize that it is a fake website in such kind of situations we don't even get the song after paying the money, so the promising solutions for such kind of problems are "The oblivious and fair data trading protocols".
In this dissertation, we have worked on one such protocol called "The new oblivious and fair data trading protocol".

Future work:

1. Need to prove the Correctness, security properties and efficiency of the New Oblivious and fair data trading protocol, need to give the comparison of the results with the other protocol.

2. Here in this problem the owner is having only two files with him but we can extend the same protocols for multiple files

3. In this protocol we have used the Zero knowledge proofs for verification of the correctness of the files but we know that for large files this computation, verification of zero knowledge proofs becomes more difficult so we need to come up with the protocols with out using the zero knowledge proofs.

4. This is a simple two party protocol but we need to extend this to multi party protocols.

Bibliography

[1] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. volume 3027, pages 506–522, 04 2004.

[2] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984, 2018.

[3] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking web applications built on top of encrypted data. pages 1353–1364, 10 2016.

[4] David J. Wu. Henry Corrigan-Gibbs, Sam Kim. Interactive proofs and zero-knowledge. pages 1–8, 2018.

[5] Peng Jiang, Xiaofen Wang, Jianchang Lai, Fuchun Guo, and Rongmao Chen. Oblivious keyword search with authorization. pages 173–190, 11 2016.

[6] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[7] Henning Pagnia, Felix C Gärtner, et al. On the impossibility of fair exchange without a trusted third party. Technical report, Citeseer, 1999.

[8] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, and Hari Balakrishnan. Building web applications on top of encrypted data using mylar. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 157–172, 2014.

[9] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International workshop on fast software encryption*, pages 371–388. Springer, 2004.