University of Illinois, Urbana Champaign
CS/ECE 498 Applied Cryptography

Author: Nishant Kumar
Date: December 16, 2020

PROJECT
1

Techniques in OT extension

In this report, we take a closer look at the primitive of Oblivious Transfer (OT). OT is a central cryptographic primitive which involves two parties - a sender and a receiver. The sender has a bunch of strings - say $\{s_1, s_2 \ldots s_n\}$ and the receiver has a selection index i and wants to retrieve the string s_i (see Figure 1.1 for a visual description). Without caring for security, this problem is trivial in that either the receiver can reveal to the sender its desired index and the sender sends the correct value to the receiver or the sender sends over all the strings $\{s_1, s_2 \ldots s_n\}$ to the receiver and it then chooses s_i. However, what makes this problem highly non-trivial is that the sender wants to send over s_i to the receiver obliviously - i.e. neither the sender learns the receiver's index i, nor the receiver learns any information other than s_i at the end of the protocol.

OT can be thought of as a special case of Secure-Multi Party Computation (MPC), which is another very well studied and powerful primitive in cryptography. Roughly speaking, MPC involves a bunch of distrustful parties with private inputs, who are interested in computing some joint function f on the combined data in such a way that only the output of the function is revealed to the parties and provably nothing else about any party's data is revealed to any other other party. A lot of practical privacy-oriented real-world computations can then be modelled in the framework of MPC. For example, consider training of Machine Learning models on private data from entities. Or performing Machine-Learning-as-a-service, where a server hosts a ML model and a client with a private input is interested in learning the ML model prediction in such a way that neither the client learns about the ML model (other than the prediction output), nor the server learns anything about the client's private data. With MPC, these problems can be solved easily.

Interestingly, OT is proven to be complete for MPC - meaning a protocol to securely evaluate OT would imply a protocol to do MPC [12, 8]. And this serves as one of the central motivations to study OT. Efficient protocols for OT would be useful to get practical real-world implementations of MPC, which as mentioned is very useful for a lot of practical privacy-oriented tasks. Other than this, theoretically speaking, its interesting in its own right to find out what the minimal cryptographic assumptions needed for OT are and if its implied by say One-way-functions.

In this report, we will look in detail at the efficiency aspect of OT. In particular, we will look at OT extension - which studies how to extend OT - i.e. use a small number of base OTs to do a large number of OTs. Impagliazzo and Rudich [9], in a seminal work

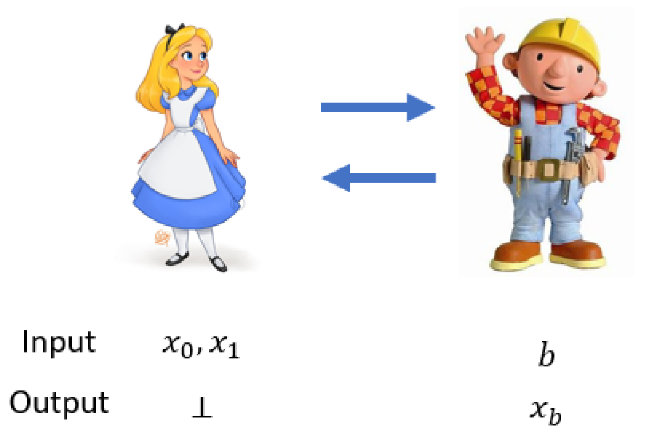


FIGURE 1.1: 1-out-of-2 Oblivious Transfer functionality

in 1989, showed that a black-box reduction from OT to a OWF would imply P$\neq$NP. This would mean that there is mostly no hope of basing OT on symmetric key primitives and in fact 2-round OT was later shown to be equivalent to 2-round key-exchange and hence public-key-encryption [7]. In light of this result, it might seem like there is little chance to make OT as efficient as symmetric key primitives (since public-key encryption is typically much slower than symmetric-key).

However, in a seminal work in 1996, Beaver [3] showed that its possible to get the next best thing - i.e. use only a small number of base OTs to extend it and get a large number of OTs using only OWF. However, the proposed construction made non-black-box use of the underlying OWF and hence was impractical for real-world usage. Ishai et al. [10] proposed another construction of OT extension in the Random Oracle Model (ROM), which only makes use of cheap symmetric key primitives, in addition to the ROM calls. Subsequent works extended the Ishai et al. [10] construction to more expressive forms of OT and increased security level with better efficiency [1, 13, 11].

This IKNP style protocols have remained the state-of-the-art (efficiency-wise) in OT extension, until very recently. In a recent series of works [5, 4, 15, 16], Ferret [16] shows how to get cheap 1-out-of-2 correlated OT (another form of OT which implies the general one, as we shall see in the next section) almost for free under variants of the LPN assumption. The idea here is that with some small communication, the 2 parties first setup small correlated seeds, which can then be "silently" expanded to long stretches of the desired correlated OT. Even more recently, Boyle et al. [6] show how to get exponential number of such correlated OTs from the Variable-density LPN assumption. This last work however, is at the moment only of theoretical interest. Despite all this exciting progress, in this report, we focus our attention on the semi-honest IKNP OT extension and explore the state-of-the-art in this style of protocols.

Notation Until otherwise mentioned, we will use the term OT to refer to 1-out-2 OT, i.e. where the sender inputs two ℓ length strings - s_0, s_1 and the receiver inputs a choice bit b and learns s_b. For ease of notation, we will also use OT_ℓ^m to denote m instances of this 1-out-of-2 OT on ℓ length strings.

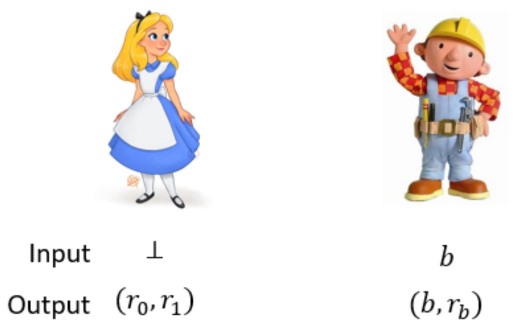


FIGURE 1.2: Random Oblivious Transfer (ROT)

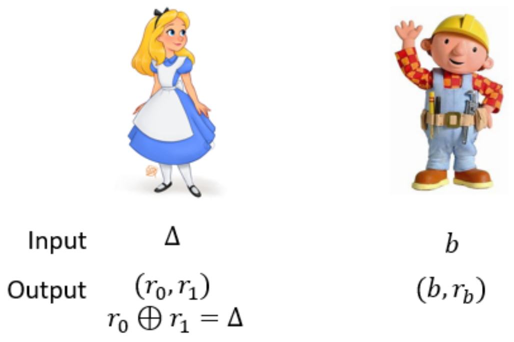


FIGURE 1.3: Correlated Oblivious Transfer (COT)

1.1 General techniques for OT

In this section, we will look at some of the general techniques which help us reduce the problem of OT_ℓ^m to something more tractable. These ideas will help us to build upto the idea of OT extension of [10], which we will see in the next section.

Random OT (ROT) In the most general form of OT, the sender has a certain set of strings and the receiver has a particular selection index. Lets call this form of OT as chosen-input OT or just OT. A simpler problem is to consider the following: the receiver inputs a selection index, and the functionality provides the sender with some set of pseudo-random strings and the corresponding chosen string (selected according to the receiver's choice index) to the receiver. Let call this form of OT as Random OT or *ROT* (see Figure 1.2 for a visual description). Also, similar to our notation for OT_ℓ^m, we use ROT_ℓ^m to denote m instances of 1-out-2 ROT on ℓ bit strings.

A bit more formally, the functionality of 1-out-of-2 ROT works as follows: the receiver, $\mathcal{R}$ inputs a choice bit b, while the sender $\mathcal{S}$ has no input. The ROT functionality samples two random strings r_0 and r_1 of length ℓ, and provides to the sender the strings r_0, r_1, while the receiver $\mathcal{R}$ gets r_b.

The important observation is that with a small amount of communication, $ROT_\ell^m \implies OT_\ell^m$. Looking ahead, this reduction will allow us to simplify our task of doing OT_ℓ^m to ROT_ℓ^m in the subsequent sections.

To see this, consider the simpler case of 1-bit strings and consider that $\mathcal{S}$ and $\mathcal{R}$ have outputs of 1-out-of-2 ROT functionality - i.e. $\mathcal{S}$ holds bits r_0, r_1 and $\mathcal{R}$ holds b, r_b. Now, $\mathcal{S}$ and $\mathcal{R}$ are interested in performing a chosen-input OT, where $\mathcal{S}$ inputs two chosen bits - (x_0, x_1) and $\mathcal{R}$ inputs the same choice bit b. With the ROT setup in place, $\mathcal{S}$ can just send over $(y_0, y_1) = (x_0 \oplus r_0, x_1 \oplus r_1)$ to $\mathcal{R}$, who does $y_b \oplus r_b$ to get x_b, as desired. A visual description of this protocol is also provided in Figure 1.4.

In the above, we assumed that $\mathcal{R}$ uses the same choice bit b in the chosen-input OT as the one used in ROT. The same can be done in fact if $\mathcal{R}$ uses a different choice bit c in the chosen-input phase, with one extra round of message. To do this, $\mathcal{R}$ first sends over $k = b \oplus c$ to $\mathcal{S}$, who permutes r_0, r_1 according to the received bit k and sends back to $\mathcal{R}$ the value $(y_0, y_1) = (x_0 \oplus r_k, x_1 \oplus r_{\bar{k}})$. $\mathcal{R}$ computes $y_c \oplus r_b = x_c$. Security follows because $\mathcal{S}$ only gets to see the masked input-bit $k = b \oplus c$. Figure 1.5 shows a visual description of

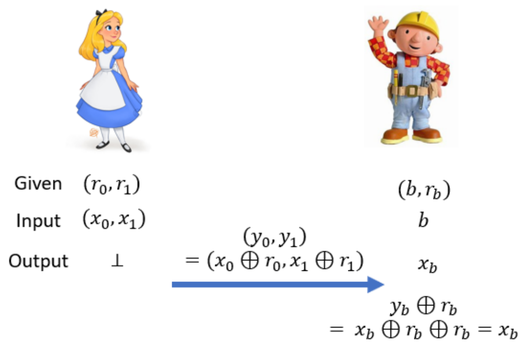


FIGURE 1.4: $ROT \implies OT$ in a single round and one message from Alice($\mathcal{S}$) to Bob($\mathcal{R}$), when the choice bit of Bob is the same across both.

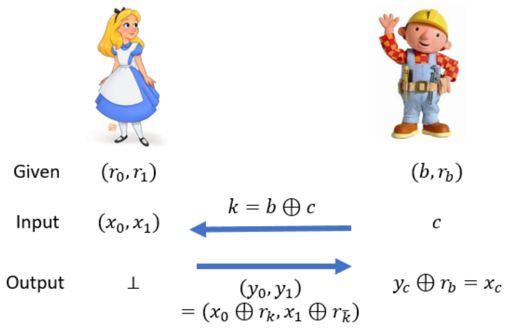


FIGURE 1.5: $ROT \implies OT$ in two rounds with a masked bit from Bob to Alice and one message from Alice($\mathcal{S}$) to Bob($\mathcal{R}$), when the choice bit of Bob can be different across both.

this protocol.

The above idea can be easily be extended to ℓ bit strings in the same manner. This idea of having preprocessed OTs (which we call ROT) is attributed to Beaver [2].

Correlated OT (COT) In the last section, we saw that OT_ℓ^m can in fact be reduced to ROT_ℓ^m with one extra message from $\mathcal{S}$ to $\mathcal{R}$ when $\mathcal{R}$ has the same choice bit in both the phases. We will now see that in the Random Oracle Model ROT_ℓ^m can be reduced to something even simpler.

Consider the following form of OT, where the sender inputs are somehow correlated with one another, but are otherwise completely random. Since we are discussing about 1-out-of-2 OT, the simplest form of correlation we will talk about is where the sender's two messages xor upto a fixed constant - say Δ. Formally, $\mathcal{S}$ inputs into this OT functionality, the correlation Δ, while $\mathcal{R}$ inputs the choice bit b. The functionality samples a random ℓ bit string r and returns to the sender the strings $(r, r \oplus \Delta)$, while $\mathcal{R}$ gets $r \oplus b\Delta$ (see Figure 1.3 for a visual description).

We will refer to this form of OT as correlated OT or COT for short. Also, in consistency with our notation introduced earlier, we will use COT_ℓ^m to denote m instances of 1-out-of-2 COT over ℓ bit strings.

ROT_ℓ^m can then be reduced further to COT_k^m, where k is the computational security parameter. This is easy to do especially given the random oracle - $\mathcal{S}$ and $\mathcal{R}$ can both just hash their respective outputs from COT_k^m using the random oracle. This also allows them to stretch their k length outputs from the random oracle to ℓ length outputs of ROT_ℓ^m.

A note on the usage of Random Oracle While the above description makes use of the random oracle, we can in fact work with something weaker, as pointed out by [10]. Roughly speaking, what we really require is the hash function to still look random even when given outputs on correlated inputs. This is what we will call as correlation robust hash function. Formally, using the definition from [10], a hash function $h : \{0,1\}^k \to \{0,1\}$ is called correlation robust if for randomly and independently chosen strings $s, t_1, t_2 \ldots t_m$, the joint distribution $(h(t_1 \oplus s), h(t_2 \oplus s) \ldots h(t_m \oplus s))$ is pseudo-random even given $t_1, t_2 \ldots t_m$. We will not explore this further in this report and leave it for future work.

1.2 The IKNP OT extension

From the last section we saw that the problem of OT_ℓ^m can be reduced to the problem of COT_k^m given the random oracle and an extra round of communication from $\mathcal{S}$ to $\mathcal{R}$. Given this insight, we will now see how to do COT_k^m given only a small number of base OTs. The technique we will be using is due to Ishai et al. [10] and is sometimes referred to as the IKNP OT-extension after the authors. The idea will be to perform COT_k^m using OT_m^k, i.e. k instances of length m OTs. Since typically total number of OTs to be performed $= m \gg k$, the security parameter, we are effectively doing a large number of OTs at the cost of a small number of OTs plus some more symmetric key and RO calls. Lets build this up in steps.

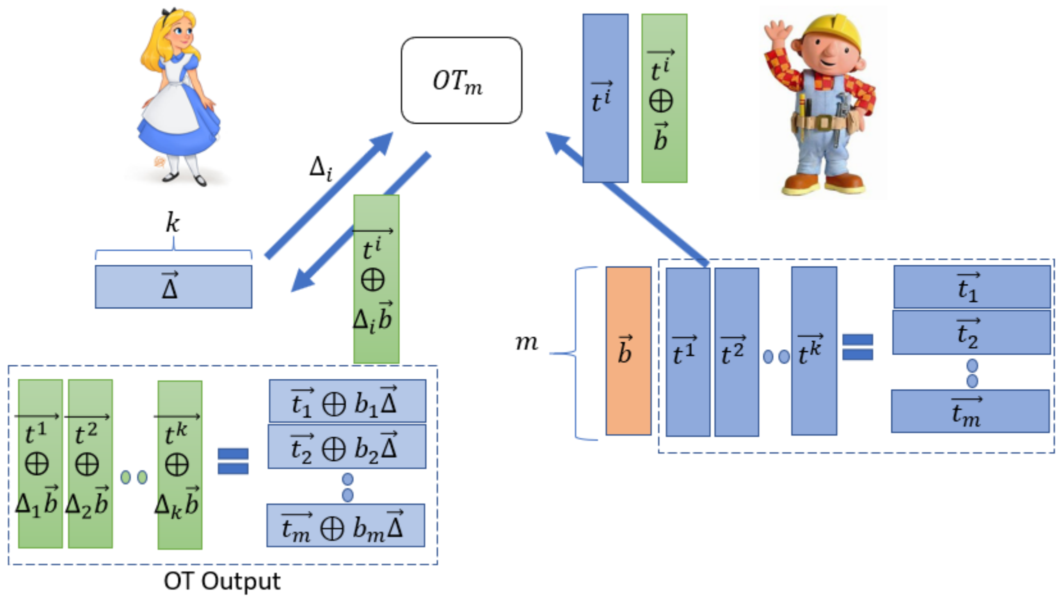
Attempt 1 Lets first try and see what our desired COT_k^m looks like. Naively, by definition of the type of correlation we are looking for, this would mean that $\mathcal{R}$ inputs into the protocol choice bits b_i and at the end, $\mathcal{S}$ gets $\{(r_i, r_i \oplus \Delta)\}$, while $\mathcal{R}$ gets $\{(b_i, r_i \oplus b_i\Delta)\}, |r_i| = |\Delta| = \ell, \forall 1 \leq i \leq m$. And we are interested in doing this using only k instaces of length m OTs, or for that matter any fewer than m OTs. The reader is encouraged to pause at this point and think how to do this. But it might seem difficult/impossible at first sight, at least given the way we are looking at this.

Attempt 2 Lets try to look at the desired correlation a bit differently. Consider that $\mathcal{R}$ inputs into the protocol choice bits b_i, but at the end $\mathcal{S}$ holds $(r_i \oplus b_i\Delta, r_i \oplus \bar{b_i}\Delta)$, while $\mathcal{R}$ holds $(b_i, r_i), \forall 1 \leq i \leq m$. Notice that this is still the same Δ-correlation - i.e. the two input strings of $\mathcal{S}$ xor upto Δ and $\mathcal{R}$ holds the chosen string. The only difference from our earlier attempt is that now $\mathcal{R}$ holds a fixed string r_i compared to holding r_{b_i} in our earlier attempt.

With this in place, we are now set to establish these correlated OTs using only OT_m^k, i.e. k instaces of length m OTs. Consider the following COT_k^m protocol: $\mathcal{S}$ chooses a correlation $\mathbf{\Delta} \in \{0,1\}^k$. Let Δ_i denote the bits of $\mathbf{\Delta}$ - i.e. $\mathbf{\Delta} = (\Delta_1 || \Delta_2 \ldots || \Delta_k)$. $\mathcal{R}$ chooses k instances of length m vectors randomly - i.e. $\mathcal{R}$ chooses $\mathbf{t^i} \in \{0,1\}^m, \forall 1 \leq i \leq k$. $\mathcal{S}$ and $\mathcal{R}$ then run OT_m^k with $\mathcal{S}$ acting as the receiver and $\mathcal{R}$ acting as the sender in the OT protocol and where $\mathcal{S}$ uses Δ_i as its choice bits, while $\mathcal{R}$ uses $(\mathbf{t^i}, \mathbf{t^i} \oplus \mathbf{b})$, as its messages. Here $\mathbf{b}$ is used to denote a length m message formed by using b_i, i.e. $\mathbf{b} = (b_1 || b_2 \ldots || b_m)$.

By the OT functionality then, $\mathcal{S}$ learns $\mathbf{t^i} \oplus \Delta_i \mathbf{b}$ as its output from the i^{th} OT. Lets now view these m length messages $\mathbf{t^i} \oplus \Delta_i \mathbf{b}, \forall 1 \leq i \leq k$ as the columns of a $m \times k$ matrix. By inspection, the rows of such of a matrix would then be $\mathbf{t_i} \oplus b_i \mathbf{\Delta}$, where we are now using $\mathbf{t_i}$ to denote the corresponding length k row vectors of the $m \times k$ matrix formed by using $\mathbf{t^i}$ as the columns. $\mathcal{S}$ can also xor this string with $\mathbf{\Delta}$ to get $\mathbf{t_i} \oplus \bar{b_i} \mathbf{\Delta}$, with which it ends up with the exact correlation we wanted in the first place - i.e. $(\mathbf{t_i} \oplus b_i \mathbf{\Delta}, \mathbf{t_i} \oplus \bar{b_i} \mathbf{\Delta}), \forall 1 \leq i \leq m$. A similar transformation done by $\mathcal{R}$ lets it put $\mathbf{t^i}$ together as the $m \times k$ matrix and then take $\mathbf{t_i}$ to denote the corresponding row vectors, with which it ends up with $(b_i, \mathbf{t_i})$.

Summary A visual description of the protocol appears in Figure 1.6. Lets try to summarize what happened. From our previous section, we reduced the problem of doing OT_ℓ^m to the problem of COT_k^m. A naive attempt to establish this correlation doesn't seem to



OT_m
Δ_i
k
$\vec{\Delta}$
$\vec{t^i} \oplus \Delta_i \vec{b}$
$\vec{t^i}$
$\vec{t^i} \oplus \vec{b}$
m
$\vec{b}$
$\vec{t^1}$
$\vec{t^2}$
$\vec{t^k}$
$\vec{t_1}$
$\vec{t_2}$
$\vec{t_m}$
$\vec{t^1} \oplus \Delta_1 \vec{b}$
$\vec{t^2} \oplus \Delta_2 \vec{b}$
$\vec{t^k} \oplus \Delta_k \vec{b}$
$\vec{t_1} \oplus b_1 \vec{\Delta}$
$\vec{t_2} \oplus b_2 \vec{\Delta}$
$\vec{t_m} \oplus b_m \vec{\Delta}$
OT Output

FIGURE 1.6: The IKNP OT extension protocol. Alice($\mathcal{S}$) chooses $\mathbf{\Delta} \in \{0,1\}^k$, while Bob($\mathcal{R}$) has input $\mathbf{b} = (b_1 || b_2 \dots || b_m)$ and chooses $\mathbf{t^i} \in \{0,1\}^m, \forall 1 \leq i \leq k$. Alice and Bob run k instances of the OT_m protocol (i.e. OT over m length strings) where Alice acts as the receiver and inputs Δ_i as the choice bit, while Bob acts as the sender and inputs $(\mathbf{t^i}, \mathbf{t^i} \oplus \mathbf{b})$. Alice as a result learns $\mathbf{t^i} \oplus \Delta_i \mathbf{b}$, which when put together as a matrix and transposed leads her to learn $\mathbf{t_i} \oplus b_i \mathbf{\Delta}$. Finally, Alice xors this with Δ to get $(\mathbf{t_i} \oplus b_i \Delta, \mathbf{t_i} \oplus \bar{b_i} \Delta)$, while Bob after similar transposition operation learns $(b_i, \mathbf{t_i}), \forall 1 \leq i \leq m$. With this Alice and Bob have now accomplished COT_k^m using OT_m^k.

be possible with better than m OTs. But viewing this correlation differently allowed us to see the original OT sender, $\mathcal{S}$, as the new OT receiver and the original OT receiver, $\mathcal{R}$, as the new OT sender. This allowed us establish the desired COT_k^m using only k instances of longer length m OTs. Once this is done, we get COT_k^m, after which $\mathcal{S}$ and $\mathcal{R}$ can locally hash it using the random oracle to get ROT_ℓ^m. Finally, $\mathcal{S}$ sends one message of size $2m\ell$ bits to $\mathcal{R}$ with which they now end up with the desired OT_ℓ^m.

One might ask at this stage, how and why does it help at all to do a large number of OTs using only a smaller number of *longer* OTs. Looking ahead, the reason is that the underlying functionality we have assumed OT_m^k can further be reduced to OT_k^k, as we shall see in the next section. But this would mean that not only are we using a smaller number of OTs, but they are also over *smaller* length strings, which is what gives us the overall efficiency advantage (since we pay the cost of public-key operations for only this small set of OT_k^k and rest is all either PRG and RO calls).

1.3 IKNP in practice (and optimizations)

Instantiating base OTs While the previous section should clear the air for the reader on how IKNP works in theory, we are still left with the task of instantiating the base OT_m^k. A simple observation allows us to reduce this OT_m^k further to OT_k^k using only a local PRG. The idea is to perform OT_k^k on seeds of a PRG of length k, which can then be locally expanded into long vectors of length m effectively yielding ROT_m^k, which can further be used to do OT_m^k with only one message from the sender to the receiver. The base OT_k^k can be instantiated with an actual public-key OT, for example, using the protocol due to Naor and Pinkas [14].

To put things together, suppose we are given OT_k^k, which can be instantiated with Naor and Pinkas' protocol for OT. In the IKNP protocol discussed previously, $\mathcal{R}$ chooses 2 seeds of a previously agreed upon PRG, G - say $(k_{i0}, k_{i1}), \forall 1 \leq i \leq k$. $\mathcal{S}$ and $\mathcal{R}$ call the OT_k^k functionality, with $\mathcal{S}$ acting as the OT receiver with choice bits Δ_i, and $\mathcal{R}$ acting as the OT sender with sender strings - $(k_{i0}, k_{i1}), \forall 1 \leq i \leq k$. With this, $\mathcal{S}$ learns $k_{i_{\Delta_i}}$. $\mathcal{S}$ and $\mathcal{R}$ now expand these PRG seeds to a length m vector, so that $\mathcal{R}$ holds $(G(k_{i0}), G(k_{i1}))$, while $\mathcal{S}$ holds $(\Delta_i, G(k_{i\Delta_i}))$. Notice that what we effectively have is ROT_m^k. Furthermore, in the IKNP protocol, in the first step, when $\mathcal{S}$ and $\mathcal{R}$ need to do perform OT_m^k, they can do this easily with one message from $\mathcal{R}$ to $\mathcal{S}$ (using the OT_m^k to ROT_m^k reduction discussed previously).

Reducing communication for Correlated OT Suppose we are interested in performing COT_ℓ^m. One way to do this is for the sender, $\mathcal{S}$ to prepare both of its messages on its own, i.e. by choosing m messages, r_{i0} of length ℓ bits each uniformly at random and then setting $r_{i1} = r_{i0} \oplus \Delta$. $\mathcal{S}$ and $\mathcal{R}$ then run OT_ℓ^m using the IKNP OT extension discussed previously. But intuitively, COT has an extra degree of freedom in choosing one of its messages pseudo-randomly and we claim that with this observation we can do better than doing OT_ℓ^m naively with the IKNP OT extension.

Recall that in the final step of the IKNP protocol, to convert ROT_ℓ^m to OT_ℓ^m, $\mathcal{S}$ needed to send $\mathcal{R}$ 2 messages per OT of length ℓ bits each, i.e. $2m\ell$ bits. Since what we care about is COT, where one of the sender's messages can be chosen pseudo-randomly, and the other message is set according to the correlation, with a clever observation, we can reduce this communication to $m\ell$ bits. This observation is attributed to [1].

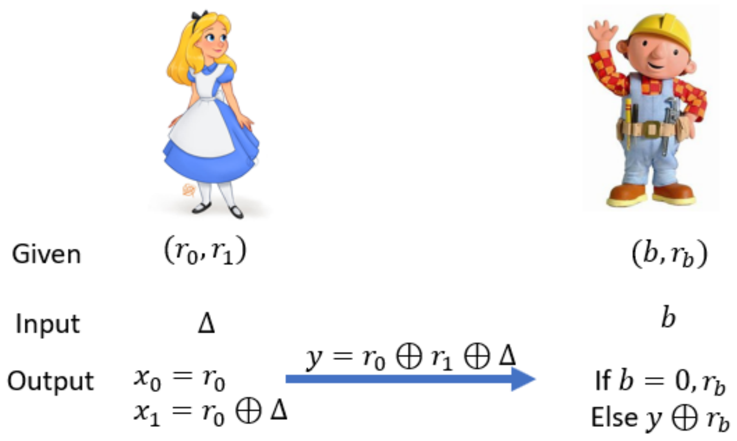


FIGURE 1.7: *ROT* $\Longrightarrow$ *COT* with a single message from Alice($\mathcal{S}$) to Bob($\mathcal{R}$) when the choice bit of the receiver remains same. This optimization allows to reduce the communication of COT and also the preprocessing OT step in the IKNP OT extension.

In particular, consider the conversion from ROT to chosen-input OT discussed earlier. $\mathcal{S}$ ad $\mathcal{R}$ are given a ROT correlation over 1 bit, i.e. $\mathcal{S}$ holds (r_0, r_1), while $\mathcal{R}$ holds (b, r_b). As discussed previously, in the chosen-input OT, if $\mathcal{S}$ has inputs (x_0, x_1), while $\mathcal{R}$ has input the same choice bit b, then $\mathcal{S}$ can send over $(y_0, y_1) = (x_0 \oplus r_0, x_1 \oplus r_1)$ and $\mathcal{R}$ performs $y_b \oplus r_b = x_b \oplus r_b \oplus r_b = x_b$.

If instead of doing this chosen-input OT, $\mathcal{S}$ has input the correlation Δ, and $\mathcal{R}$ has input the same choice bit b, they can perform the following protocol: $\mathcal{S}$ sends over $r_0 \oplus r_1 \oplus \Delta$ and outputs $(m_0, m_1) = (r_0, r_0 \oplus \Delta)$. $\mathcal{R}$ on receiving $y = r_0 \oplus r_1 \oplus \Delta$ outputs $r_b = r_0 = m_0$ if $b = 0$, else outputs $r_b \oplus y = r_1 \oplus r_0 \oplus r_1 \oplus \Delta = r_0 \oplus \Delta = m_1$.

And observe that we only communicated a single bit from $\mathcal{S}$ to $\mathcal{R}$, while in the previous chosen-input case, we communicated 2 bits from $\mathcal{S}$ to $\mathcal{R}$. This protocol is also shown figuratively in Figure 1.7.

Lets try to intuitively understand what is happening. $\mathcal{S}$ and $\mathcal{R}$ held one instance of ROT correlation - i.e. $\mathcal{S}$ held (r_0, r_1), while $\mathcal{R}$ held (b, r_b). Now, in the COT phase, $\mathcal{S}$ has input correlation Δ, while $\mathcal{R}$ has input the same choice bit b. If we take it that the messages output by $\mathcal{S}$ at the end are $(m_0, m_1) = (r_0, r_0 \oplus \Delta)$, then if $b = 0$, $\mathcal{R}$ needs to output r_0, but note that it already holds $r_b = r_0$. In case $b = 1$, $\mathcal{R}$ needs to output $r_0 \oplus \Delta$ and it holds r_1. $\mathcal{S}$ can send over then $y = r_0 \oplus r_1 \oplus \Delta$, which allows $\mathcal{R}$ to unmask r_1 and recover $r_0 \oplus \Delta$. Also, note that in case $b = 0$, in sending over $r_0 \oplus r_1 \oplus \Delta$, $\mathcal{R}$ doesn't know r_1 and hence doesn't get any more information from the incoming message (since r_1 essentially works as as masking factor).

An important point to note is that the above conversion from ROT to COT with lesser communication is possible when the choice bit of the receiver is same across both the instances. Also, the same technique can be extended to work with ℓ bit messages, which is why the commnication from $\mathcal{S}$ to $\mathcal{R}$ in the second step of IKNP is halved when dealing with COT rather than chosen-input OT.

Putting it all together Recall from the first step of the IKNP protocol, we needed to do an OT with $\mathcal{S}$ acting as the receiver with choice bit Δ_i, while $\mathcal{R}$ acted as the sender with input messages $(\mathbf{t^i}, \mathbf{t^i} \oplus \mathbf{b})$, where $\mathbf{t^i}$ is chosen randomly $\forall 1 \leq i \leq k$. But note that this is in fact a COT that is happening! And which means we can use our observation from the previous paragraph to cut down the communication of this step from $\mathcal{R}$ to $\mathcal{S}$ by half.

This sums up all the optimizations that are used in practice to implement the IKNP protocol, to the best of the report's author's knowledge.

Efficiency Ignoring the cost of the base OT protocol, the cost of the first message from $\mathcal{R}$ to $\mathcal{S}$ is mk (after applying the last optimization this first step is in fact a COT rather than OT, as discussed previously and which is why the cost is mk and not $2mk$). Assuming only a chosen-input OT, the second step involves a communication of $2m\ell$ bits from $\mathcal{S}$ to $\mathcal{R}$, making the total communication to be of $m(k + 2\ell)$ (for OT_ℓ^m). In addition, if what we are interested in is COT_ℓ^m, then the cost is $m(k + \ell)$, while if are interested in ROT_ℓ^m, then the cost is mk. Note that other than communication, the other major cost is that of computation, which mainly involes two parts - the transposition operation done by $\mathcal{S}$ and $\mathcal{R}$ and the RO calls made by them. The former has been optimized down to cache-level in [1], while the latter is optimized by making use of fixed-key AES calls. We leave details of both of these for future reports.

1.4 Conclusion

We thus have seen the classic IKNP style OT extension protocols. This comes in multiple flavous and we saw how the cost of OT_ℓ^m is $m(k + 2\ell)$, while that of COT_ℓ^m and ROT_ℓ^m are $m(k + \ell)$ and $m\ell$ respectively (ignoring the cost of the base OT protocol). Though the 1-out-of-2 semi-honest IKNP protocol we saw is no longer the state-of-art (superseeded by the silent OT line of work, in particular, by Ferret [16]), the lessons learnt from this class of protocols are important and useful to learn.

References

[1] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548, 2013.

[2] D. Beaver. Precomputing oblivious transfer. In *Annual International Cryptology Conference*, pages 97–109. Springer, 1995.

[3] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488, 1996.

[4] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round ot extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 291–308, 2019.

[5] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *Annual International Cryptology Conference*, pages 489–518. Springer, 2019.

[6] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density lpn. Cryptology ePrint Archive, Report 2020/1417, 2020. https://eprint.iacr.org/2020/1417.

[7] Y. Gertner, S. Kannan, T. Malkin, O. Reingold, and M. Viswanathan. The relationship between public key encryption and oblivious transfer. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 325–335. IEEE, 2000.

[8] O. Goldrcich and R. Vainish. How to solve any protocol problem-an efficiency improvement. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 73–86. Springer, 1987.

[9] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 44–61, 1989.

[10] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*, pages 145–161. Springer, 2003.

[11] M. Keller, E. Orsini, and P. Scholl. Actively secure ot extension with optimal overhead. In *Annual Cryptology Conference*, pages 724–741. Springer, 2015.

[12] J. Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31, 1988.

[13] V. Kolesnikov and R. Kumaresan. Improved ot extension for transferring short secrets. In *Annual Cryptology Conference*, pages 54–70. Springer, 2013.

[14] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, volume 1, pages 448–457, 2001.

[15] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-ole: Improved constructions and implementation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1055–1072, 2019.

[16] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. Ferret: Fast extension for correlated ot with small communication. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1607–1626, 2020.