Oblivious Transfer Extension

David Heath

Fall 2025

At this point, we have seen two main techniques for achieving MPC of arbitrary programs, one based on secret sharing (the GMW protocol), and one based on garbled circuits. Last time we even saw how to generalize garbled circuits to work for any number of parties (the BMR protocol).

In all of the protocols we have seen, a main ingredient has been the *extensive* use of Oblivious Transfer (OT). Indeed, the GMW and BMR protocols both need $O(p^2 \cdot |C|)$ OTs, so even relatively modest computations with a small number of parties can easily consume thousands or even millions of OTs.

Remark 1 (2PC garbled circuits use fewer OTs)**.** *The two-party garbled circuit technique still requires OT, but it uses significantly less. In particular, for that protocol, we needed only $O(n)$ OTs, where n is the size of the* evaluator's input *to the desired computation. Indeed, we do not even need OTs for the garbler's input. That is, the number of required OTs does not scale with the total circuit size; this fact will be important for today's discussion.*

As we have discussed, OT can be constructed from certain public-key encryption schemes. Recall that semi-honest 1-out-of-2 OT can be constructed as follows:

- The receiver samples two public keys $\texttt{pk}_s$ and $\texttt{pk}_{1-s}$, where $b \in bit$ is the receiver's choice bit. Public key $\texttt{pk}_s$ is sampled in such a way that the receiver also holds a matching key $\texttt{sk}_s$, but $\texttt{pk}_{1-s}$ is simply chosen at random, without a matching secret key.
- The receiver sends public keys $\texttt{pk}_0, \texttt{pk}_1$ to the sender, in that order.
- The sender encrypts its two messages m^0, m^1 with the public keys and sends back two ciphertexts $c_0 = \texttt{Enc}(\texttt{pk}_0, m^0)$ and $c_1 = \texttt{Enc}(\texttt{pk}_1, m^1)$.
- Since the receiver has only one matching secret key $\texttt{sk}_s$, she can only decrypt message m^b, and not m^{1-s}.

Of course, this protocol requires calling public-key procedures `Enc` and `Dec`, and it requires transmission of public keys. As described so far in this course, the parties invoke the above protocol $O(p^2 \cdot |C|)$ total times to complete the BMR/GMW protocol.

The problem with this is that public-key encryption schemes are quite expensive. As one example of this expense, the nearly-ubiquitous RSA encryption scheme has public keys that are *thousands* of bits long. This means that the techniques we have discussed so far are essentially impractical. However, if we could greatly improve the cost of OT, these techniques would become vastly more efficient.

Symmetric-key encryption schemes are generally very significantly more efficient than their public-key counterparts. This should not be surprising, as the problem solved by symmetric-key schemes is *strictly easier* than that of public-key schemes. Therefore, we can get away with less powerful cryptographic mechanisms. Indeed, it is often fair to assume that a symmetric-key encryption scheme is *orders of magnitude* more efficient than a public-key encryption scheme. As an example, the ubiquitous AES scheme often has keys of length 128 bits, dozens of times shorter than RSA keys.

Therefore, there is a clear route to pursue: implement OT with a symmetric-key encryption scheme, rather than a public-key one. Unfortunately, we have very strong evidence that this is impossible [IR90]. Namely, we believe that OT cannot be achieved with symmetric-key cryptography alone.

However, there is a way to circumvent this difficulty: we have strong evidence that public-key cryptography is necessary to achieve OT, but this says nothing about *how much* public key cryptography is required. In particular, suppose we wish to prepare a huge number of OTs n; so far, our techniques require $O(n)$ public-key operations. Perhaps we can use far fewer, say $O(\lambda)$.

Today we will see that this is indeed possible. We will investigate the OT extension technique. In short, OT extension *extends* a small batch of $O(\lambda)$ OTs into a much larger batch of $O(n)$ OTs. While we need public-key operations to prepare the so-called *base* OTs, the larger batch can be prepared using only symmetric-key operations.

An Analogy to Hybrid Encryption

The OT extension technique is analogous to an extremely common technique in secure communication. When Alice and Bob wish to securely communicate on the internet, they can initiate their conversation by using public-key encryption techniques. However, typically they do not send their actual messages with public-key encryption. Again, public-key encryption is expensive, and we would like to avoid it.

Instead, Alice and Bob typically would use public-key encryption to exchange only a single, small message: a symmetric encryption key. From here, they can use this symmetric key to far more efficiently encrypt their messages. This technique is typically called *hybrid encryption*.

OT extension leverages the same main idea: the parties use a bare minimum number of public-key operations to exchange key material, then they use symmetric-key techniques to exchange their desired messages.

Random OT

Recall from when we originally defined OT that there are many OT variants. Formally, today we will consider both the standard, chosen-message OT, as well as *random* OT. We will define the random OT functionality as follows:

- The sender and receiver agree on a message length ℓ.
- The functionality uniformly samples two messages $m^0, m^1 \in_\$ \{0,1\}^\ell$ and a choice bit $b \in_\$ \{0,1\}$.
- The functionality sends m^0, m^1 to the sender, and it sends b, m^b to the receiver.

This is the functionality we will implement. Note that random OT can be easily transformed into chosen-message OT.

OT Length Extension

Our goal is to transform a small number of OTs into a large number of OTs. However, to do this efficiently, we will eventually need a much simpler transformation that converts a *single* random OT of relatively small messages into a single random OT of much larger messages.

This is notion of OT *length extension* is achieved in a straightforward manner: the parties use OT to exchange pseudorandom generator seeds, then they locally expand those to obtain the random messages.

- The sender and receiver invoke the random OT functionality with length λ. The sender therefore obtain PRG seeds k_0, k_1; the receiver obtains b and k_s.
- The sender sets $m^0 \leftarrow G(k_0)$ and $m^1 \leftarrow G(k_1)$. The receiver sets $m^s \leftarrow G(k_s)$.

This approach is clearly correct, and it is secure by the PRG: Since she does not know k_{1-s}, to an adversarial receiver the message m^{1-s} looks random.

Beaver's OT Extension

Beaver was the first to demonstrate an OT extension construction[Bea96]. His is a feasibility result, but it is quite inefficient, since it is non-black-box.

The idea is a relatively simple application of garbled circuits:

- The receiver chooses a short PRG seed $k \in \{0,1\}^\lambda$. She defines her n choice bits as $s \leftarrow G(k)$.
- From here, the sender prepares a garbled circuit, into which he will feed his $2n$ messages and the receiver will feed her seed k. The garbled circuit will then (1) expand $s \leftarrow G(k)$, then for each pair of messages m_i^0, m_i^1 and choice bit s_i it will compute $m_i^{s_i}$ and output the result.
- The parties use this garbled circuit to run the standard semi-honest garbled circuit 2PC protocol. **Note that only λ OTs are required.** This is because the evaluator's (i.e., the receiver's) input is a length-λ PRG seed.

The approach is clearly correct and secure, due to security of G and the 2PC protocol.

Unfortunately, while it establishes an important feasibility result, this approach is highly expensive, due to its non-black-box use of the PRG G (the garbled circuit includes a call to G).

IKNP OT Extension

One of the most significant developments in MPC was the discovery of the IKNP OT extension technique [IKNP03]. This technique achieves OT extension from black-box cryptography, and it was transformative to the efficiency of our techniques.

To understand this technique, we will work in the *random oracle model.*

Definition 1 (Random oracle model)**.** *In the random oracle model, we assume all parties have access to a hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$. We model the behavior of H as a trusted third party that implemented a truly random function.*

It's worth briefly stating that random oracles (ROs) are objects of some contention. Indeed, we know that random oracles are *uninstantiable* [CGH98]. Still, RO is an incredibly useful heuristic for designing new cryptography [BR93], and in this course we will often use it unapologetically.

Now, let us return to the problem of designing OT extension. The main idea of IKNP is to let the receiver choose a random choice string $s \in \{0,1\}^n$ and to let the sender choose a random *correlation* $\Delta \in \{0,1\}^\lambda$. Note that Δ is very short in comparison to the number of OTs n. Now, we wish to allow the parties to somehow compute XOR secret shares of the $n \times \lambda$ matrix $[s \otimes \Delta]$.

Note that the matrix $s \otimes \Delta$ has the following form:

$$\begin{bmatrix} s_0 \cdot \Delta_0 & s_0 \cdot \Delta_1 & \dots & s_0 \cdot \Delta_{\lambda-1} \\ s_1 \cdot \Delta_0 & s_1 \cdot \Delta_1 & \dots & s_1 \cdot \Delta_{\lambda-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n-1} \cdot \Delta_0 & s_{n-1} \cdot \Delta_1 & \dots & s_{n-1} \cdot \Delta_{\lambda-1} \end{bmatrix}$$

There are two basic observations worth observing about this matrix:

- Each i-th *row* of the matrix is equal to Δ scaled by the bit s_i.
- Each j-th *column* of the matrix is equal to s scaled by the bit Δ_j.

Now, recall that our goal is to construct *secret shares* of this matrix. Suppose we achieve this task; we will see how shortly. Now, consider the i-th row, and let us suppose the sender's share of that row happens to be $R_i \in \{0,1\}^\lambda$. Since that row is $s_i \cdot \Delta$, we know that there are only two possibilities:

- s_i is equal to zero. In this case, the row is equal to zero, so the two parties must hold equal shares. Namely, the receiver holds R_i.
- s_i is equal to one. In this case, the row is equal to Δ, so the party shares must differ by exactly Δ. Namely, the receiver holds $R_i \oplus \Delta$.

This is essentially OT. There are two possible message R_i and $R_i \oplus \Delta$. The sender indeed knows them both, since he holds R_i and Δ, and the receiver holds exactly one of them, according to her bit s_i.

Of course, this applies *for every row of the matrix.* Thus, the parties indeed hold n random OT-style correlations. The only problem is that this all rows of the matrix involve the same difference Δ. This is where we invoke the random oracle. In particular, for each row i, the parties set their outputs as follows:

- The sender sets $m^0 = H(i, R_i)$ and $m^1 = H(i, R_i \oplus \Delta)$.
- The receiver sets $m^{s_i} = H(i, R_i \oplus s_i \cdot \Delta)$.

This use of the random oracle breaks correlations between the rows. Intuitively, the approach is secure, since the receiver cannot obtain any message m^{1-s_i} except by guessing Δ. Since Δ is λ bits long, this is infeasible.

Thus, if the parties can arrange that they hold secret shares of the matrix $[s \otimes \Delta]$, then they can indeed generate n random OTs.

To compute this matrix, the parties will use λ public-key-based OTs on the *columns* of the matrix. The main surprise here is that the parties will *swap roles* to construct the matrix. Namely, to construct the matrix, our OT receiver will play the role of base OT sender; our OT sender will play the base OT receiver. This makes sense: For each *column* j of the matrix, there are two possibilities:

- Δ_j is equal to zero. In this case, the parties should obtain secret shares of zero.
- Δ_j is equal to one. In this case, the parties should obtain secret shares of s.

Thus, for each *column* of the matrix, the OT receiver (base OT sender) first samples a random string $r \in_\$ \{0,1\}^n$, and she prepares to base OT messages: $m^0 = r$ and $m^1 = r \oplus s$. She sets her share of the j-th column as r. The OT sender (base OT receiver) sets his choice bit to Δ_j, and sets his share as his base OT output. If Δ_j is equal to zero, his share indeed matches the receiver's, so they hold secret shares of zero; otherwise, his share differs from the receiver's by s, so they hold secret shares of r.

By performing this step on every column of the matrix, the parties indeed obtain $[s \otimes \Delta]$ which, as already described, they can transform into n OT correlations. To restate, the full IKNP protocol is as follows:

- The receiver samples a random choice string $s \in_\$ \{0,1\}^n$; the sender samples a correlation $\Delta \in_\$ \{0,1\}^\lambda$.
- The parties swap roles and use λ base OTs to obtain secret shares of the columns of the matrix $s \otimes \Delta$.
- The parties then hash their shares of each row of the matrix.

The result is n OT correlations. Notice that this only requires $O(\lambda)$ public-key operations.

Next Time

We now have a large suite of MPC tools, and we have even substantially improved efficiency. However, all of our techniques so far assume that the adversary follows the protocol as we describe it. What if the adversary tries to cheat? Next time, we will begin studying maliciously secure protocols, which are resilient against attempts to cheat. In particular, we will first examine zero knowledge proofs, and show connections between semi-honest and malicious protocols.

References

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Berlin, Heidelberg, August 2003.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 8–26. Springer, New York, August 1990.