

Oblivious Transfer Extension

David Heath

Fall 2026

In all of the secure computation protocols we have so far seen, a main ingredient has been a heavy utilization of Oblivious Transfer (OT). Indeed, the secret-sharing-based GMW protocol and garbled-circuit-based BMR protocol 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 fewer. 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 $\mathbf{pk}_s$ and $\mathbf{pk}_{1-s}$, where $s \in \{0, 1\}$ is the receiver's choice bit. Public key $\mathbf{pk}_s$ is sampled in such a way that the receiver also holds a matching key $\mathbf{sk}_s$, but $\mathbf{pk}_{1-s}$ is simply chosen at random, without a matching secret key.
- The receiver sends public keys $\mathbf{pk}_0, \mathbf{pk}_1$ 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 $c_0 = \text{Enc}(\mathbf{pk}_0, m_0)$ and $c_1 = \text{Enc}(\mathbf{pk}_1, m_1)$.
- Since the receiver has only one matching secret key $\mathbf{sk}_s$, she can only decrypt message m_s , 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 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 costly. However, if we could greatly improve the cost of OT, these techniques would become significantly more efficient.

Symmetric-key encryption schemes are generally 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. In particular, we will study the notion of *OT extension*. 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 a 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, so we would like to avoid it to the extent possible.

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. Today we will consider both standard chosen-message OT and *random* OT. One might naively define random OT as a functionality that samples $m_0, m_1 \leftarrow \{0, 1\}^\ell$ and $b \leftarrow \{0, 1\}$ uniformly and hands (m_0, m_1) to the sender and (b, m_b) to the receiver. This definition is too strong for our purposes: every protocol we see today derives OT messages by locally expanding a seed or hashing a share, so a corrupt party's messages will be *pseudorandom*, not uniform, and no simulator can make a uniformly sampled string lie in the image of a PRG. We therefore adopt the *endemic* variant [MR19], in which the functionality samples only the honest parties' values and lets the adversary choose the corrupt party's values.

Definition 1 (Endemic random OT). *The sender and receiver agree on a message length ℓ .*

- *If the sender is corrupt, the adversary supplies $m_0, m_1 \in \{0, 1\}^\ell$. The functionality samples $b \leftarrow \{0, 1\}$ and sends (b, m_b) to the receiver.*
- *If the receiver is corrupt, the adversary supplies $b \in \{0, 1\}$ and $m_b \in \{0, 1\}^\ell$. The functionality samples $m_{1-b} \leftarrow \{0, 1\}^\ell$ and sends (m_0, m_1) to the sender.*
- *If both parties are honest, the functionality samples m_0, m_1, b uniformly and sends (m_0, m_1) to the sender and (b, m_b) to the receiver.*

Endemic random OT is exactly as useful as the naive notion: it converts to chosen-message OT by the standard derandomization.

OT Length Extension

Our goal is to transform a small number of OTs into a large number of OTs. To do this efficiently, we will first need a simpler transformation that converts a single random OT of short messages into a single random OT of longer messages. This notion of *OT length extension* is straightforward: the parties use OT to exchange PRG seeds, then locally expand those seeds. Let $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^\ell$ be a PRG.

- The parties invoke endemic random OT with message length λ . The sender obtains seeds k_0, k_1 ; the receiver obtains b and k_b .
- The sender outputs $m_0 := G(k_0)$ and $m_1 := G(k_1)$. The receiver outputs b and $m_b := G(k_b)$.

Correctness is immediate.

Exercise 1. *Prove the above protocol is a secure instantiation of endemic random OT.*

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 uses non-black-box cryptography.

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 $s := G(k)$.
- 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 expand $s := G(k)$, then for each pair of messages $m_{i,0}, m_{i,1}$ and choice bit s_i it will compute m_{ik,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 clearly achieves a notion of OT, and it is secure due to properties of G and the 2PC protocol.

Exercise 2 (Beaver's OT extension is endemic). *Show Beaver's OT extension protocol is secure. In particular, let $G : \{0, 1\}^\lambda \rightarrow \{0, 1\}^n$ be a PRG, and let $\mathcal{F}_{2PC}$ be the ideal secure 2PC functionality (instantiable from base OTs and garbled circuits). Prove that Beaver's OT extension protocol securely realizes n instances of endemic random OT (Definition 1) against a semi-honest adversary in the $\mathcal{F}_{2PC}$ -hybrid model.*

Unfortunately, Beaver's protocol establishes an important feasibility result, it is also 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. In particular, some back-of-the-napkin math shows that IKNP improved the runtime cost of semi-honest OT by roughly 1000 $\times$, and it also modestly improves bandwidth consumption compared to the leanest public-key-based OT. This was transformative, because it shifted the cost of MPC almost entirely to *communication*, as OTs were suddenly almost free in terms of computation; see also [ALSZ13].

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

Definition 2 (Random oracle model). *In the random oracle model, we assume all parties have access to a hash function $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$. We model the behavior of H as a trusted third party that implements 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's choice bits be a string $s \in \{0, 1\}^n$, and to let the sender hold 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 noting 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 messages 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 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 j , the parties proceed as follows. They invoke base random OT with length λ , with roles swapped: our OT receiver acts as base-OT sender and obtains seeds $k_{j,0}, k_{j,1}$; our OT sender acts as base-OT receiver with choice bit Δ_j and obtains k_{j,Δ_j} . The OT receiver sets her share of column j to $r_j := G(k_{j,0}) \in \{0, 1\}^n$ and sends the single n -bit correction

$$u_j := G(k_{j,0}) \oplus G(k_{j,1}) \oplus s.$$

If $\Delta_j = 0$, the OT sender sets his share to $G(k_{j,0}) = r_j$, so the parties hold shares of zero. If $\Delta_j = 1$, he sets his share to $G(k_{j,1}) \oplus u_j = r_j \oplus s$, so the parties hold shares of s . The OT sender learns nothing about s from u_j , since it is masked by $G(k_{j,1-\Delta_j})$, whose seed he does not hold.

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 parties swap roles and use λ base OTs of λ -bit seeds, plus $n\lambda$ bits of communication, to obtain secret shares of the columns of $s \otimes \Delta$.
- The parties hash their shares of each row of the matrix.

The result is n (endemic) random OTs, using only $O(\lambda)$ public-key operations and $O(n\lambda)$ communication.

References

- [ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.
- [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.
- [MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, November 2019.