CS548: Cryptography II

Final Research Paper

On

Protocols for Secure 1-2 Oblivious Transfer

Abstract Protecting data in use is a key area of study in modern cryptography. This paper provides a description of the complexities of providing security to data that is being actively used in computation, and outlines solutions that have been discovered to more effectively secure such data. In particular, this paper focuses on protocols for one out of two oblivious transfer (1-2 OT). Two possible protocol specifications (Rivest 1999 and Even, Goldreich, and Lempel 1985), for 1-2 OT are discussed in detail, including proofs of their correctness and security. Additionally, a method for extending 1-2 OT to 1-N OT is discussed. Implementations and descriptions of the discussed protocols are also implemented in Python to serve as a demonstration of how each protocol works.

Repository https://github.com/reecetyl/CS548-Final-Paper

Tyler Reece
Boston University
College of Arts and Sciences
Boston, Massachusetts
December 2020

1 Introduction and Background

Oblivious transfer was originally introduced by Michael Rabin in 1981 as a method for a sender to transfer a piece of information to a receiver, without the sender knowing which piece of information was transferred [6]. Oblivious transfer is vitally important because it is the sole cryptographic primitive needed to implement secure multiparty computation (sMPC).

Secure multiparty computation was first conceptualized by Andrew Yao in 1986 [3]. sMPC attempts to allow several potentially distrusting parties to jointly compute a function while keeping each party's individual inputs private. From securing databases to performing computations on sensitive medical or financial data, sMPC provides a promising potential for cooperation where data is otherwise too valuable to be shared without protection [10]. While this paper will narrow its focus to the problem of oblivious transfer, it is important to note that oblivious transfer is the foundation for these more valuable technologies.

For the purposes of this paper, I will focus on "one out of two" oblivious transfer, or simply 1-2 OT. Such protocols take place under the following circumstances: the sender (whom I will call Alice), possesses two pieces of information (m_0, m_1), and the receiver (whom I will call Bob), possesses a **selection bit**, $b \in \{0, 1\}$, which denotes the message he desires to obtain.

An oblivious transfer protocol is said to be **correct** if the protocol successfully transfers m_b from Alice ($\mathcal{A}$) to Bob ($\mathcal{B}$). More formally, we say that for all inputs $m_0, m_1 \in \{0, 1\}^*$,

$$\Pr[\text{output}_{\mathcal{B}}\langle \mathcal{A}(m_0, m_1), \mathcal{B}(b)\rangle = m_b] = 1$$

An oblivious transfer protocol is said to be secure if it satisfies both **sender privacy** and **receiver privacy**. Sender privacy is the property that after the protocol is completed, Bob only learns his desired message m_b, and learns nothing about the other message m_{1-b}. More formally, there exists an efficient simulator $\text{Sim}_{\mathcal{B}}$ so that for all messages (m_0, m_1) and $b \in \{0, 1\}$,

$$\text{view}_{\mathcal{B}}((m_0, m_1), b) \approx_c \text{Sim}_{\mathcal{B}}(b, m_b)$$

This equation formally summarizes the property that the transfer should occur in such a way that from Bob's perspective, Bob learns nothing more than he could have learned if he simply ran a simulation of the protocol with self-generated inputs.

Receiver privacy is the property that after the completion of the protocol, Alice does not learn anything about Bob's selection bit, b. More formally, there exists an efficient simulator $\text{Sim}_{\mathcal{A}}$ so that for all messages (m_0, m_1) and $b \in \{0, 1\}$,

$$\text{view}_{\mathcal{A}}((m_0, m_1), b) \approx_c \text{Sim}_{\mathcal{A}}(m_0, m_1)$$

Similar to the previous equation, this summarizes the property that the transfer occurs in such a way that from Alice's perspective, she learns nothing more than if she simply ran a simulation of the protocol with her own inputs.

There are usually two threat models that are considered when discussing OT. These deal with the security of the protocol when dealing with (1) *semi-honest* (sometimes referred to as honest-but-curious) or (2) outright *malicious* parties. In the semi-honest case, both parties are assumed to follow the protocol specification exactly, sampling randomness fairly and doing all computations as specified, and only afterwards use whatever information they have to learn about the other party's private input. In the malicious case, both parties can deviate from the protocol specification at any point in order to trick the other party into revealing more information than the protocol would otherwise allow. For the purposes of this paper, I focus on the honest-but-curious party case for security.

2 Rivest's 1-2 OT Protocol

One simple protocol for 1-2 OT was given by Ron Rivest in 1999 [1]. Below, I give an outline of the original protocol, including a description and proof of correctness and security. Afterwards, I discuss an expansion in the analysis of security for a particular case, and show how the protocol could be expanded to require only one intervention from the trusted initializer before completing unboundedly many transfers.

This protocol requires the use of a "trusted initializer" (TI; which I will refer to as Ted), a third party who sets initial conditions before the protocol proceeds. This protocol uses the one-time pad (OTP), as sensitive information in the protocol is XORed with random secret values to protect the information from other parties. The protocol is illustrated below, where $*$ denotes lines given to the party by the trusted initializer .

Figure 1: 1-2 OT using TI and OTP

Alice		**Bob**
$m_0, m_1 \in \{0,1\}^k$		$b \in \{0,1\}$
$* \, r_0, r_1 \in \{0,1\}^k$		$* \, d \in \{0,1\}, r_d$
		$e = b \oplus d$
	$\xleftarrow{\quad e \quad}$	
$f_0 = m_0 \oplus r_e$		
$f_1 = m_1 \oplus r_{1-e}$		
	$\xrightarrow{\quad f_0, f_1 \quad}$	
		$m_b = f_b \oplus r_d$

Description Alice begins with two messages (m_0 and m_1) of length k bits while Bob begins with a **selection bit**, b which denotes which of the two messages he wishes to receive. The trusted initializer, Ted, randomly generates two pads r_0 and r_1 of length k, and gives them to Alice. Ted also generates a random bit d, and gives d and the corresponding pad, r_d to Bob. At this point, Ted is no longer necessary for the remainder of the protocol to proceed. Bob computes $e = b \oplus d$, thereby masking the selection bit from Alice. Alice computes f_0 and f_1 using the OTP with the corresponding r pads as keys, and sends this pair to Bob, who is then able to recover the desired message m_b through his knowledge of r_d.

Correctness An illustration of the correctness of the protocol is shown below, beginning with the final line of the algorithm and working upwards:

$$\begin{aligned} m_b &= f_b \oplus r_d \\ &= (m_b \oplus r_e) \oplus r_d \\ &= (m_b \oplus r_d) \oplus r_d \\ &= m_b \end{aligned}$$

Security The security proof for both parties in this protocol reduces to the security proof of OTP. We can see that the privacy of the selection bit, b of the receiver is protected via its XOR with d. Under the assumption that d is securely randomly generated by Ted, e is uniformly distributed over $\{0,1\}$, and therefore:

$$\Pr[\text{Sender guesses } d] = \Pr[\text{Sender guesses } b] = \tfrac{1}{2}$$

The confidentiality of the unchosen message, m_{1-b}, is protected because Bob only knows the value r_d given by Ted, and does not know the value r_{1-d}. As long as the value r_{1-d} is chosen randomly from the set $\{0,1\}^k$, the probability of recovering m_{1-b} equals the probability of guessing r_{1-d} and the following identity holds:

$$\Pr[\text{Receiver obtains } m_{1-b}] = \Pr[\text{Receiver guesses } r_{1-d}] = \tfrac{1}{2^k}$$

which is negligible for sufficiently large values of k.

Even, Goldreich, Lempel 1-2 OT Protocol

While the use of a third party trusted initializer may work for some applications, situations often arise where a lack of trust or acceptable intermediary necessitates a protocol that only involves the two communicating parties. In such situations, protocols that make use of public key cryptography are extremely useful. One such protocol that leverages the public key cryptography ideas was created by Even, Goldreich, and Lempel (EGL) in 1985 [2]. A generalized variant of EGL utilizing the Decisional Diffie-Hellman (DDH) assumption is shown below.

Figure 2: EGL 1-2 OT based on DDH

Alice		**Bob**
m_0, m_1		$b \in \{0,1\}$
		choose r $\in Z_q$
		if b $= 0 : h_0 = g_0^r, h_1 = g_1^r$
		if b $= 1 : h_0 = g_0^r, h_1 = g_1^{r+1}$
	$\xleftarrow{h_0, h_1}$	
$a_0, b_0, a_1, b_1 \leftarrow Z_q$		
	$c_0 = \xrightarrow{(g_0^{a0} g_1^{b0}, h_0^{a0} h_1^{b0} m_0)}$	
	$c_1 = \xrightarrow{(g_0^{a_1} g_1^{b_1}, h_0^{a_1} h_1 / g_1^{b_1} m_1)}$	
		$c_b = (c^1, c^2)$
		$m_b = c^2/(c^1)^r$

Description In the protocol above, Alice begins with two messages, m_0 and m_1. Bob holds a selection bit, $b \in \{0,1\}$, and samples a random r from Z_q, and computes h_0, h_1 as shown and sends them to Alice. Alice generates four new values from Z_q, and uses them to encrypt both m_0 and m_1 which are sent to Bob. Bob uses his selection bit to choose a ciphertext, and parses it as (c^1, c^2). He finally uses the properties of DDH to recover the desired message m_b.

Correctness We can show (similar) proofs of correctness for each of the two cases, $b = 0$ and $b = 1$. For each proof, we start at the end of the protocol and work upwards. For $b = 0$, we have:

$$\begin{aligned}
m_0 &= c^2/(c^1)^r \\
&= \frac{h_0^{a_0} h_1^{b_0} m_0}{(g_0^{a_0} g_1^{b_0})^r} \\
&= \frac{(g_0^r)^{a_0} (g_1^r)^{b_0} m_0}{((g_0)^{a_0} (g_1)^{b_0})^r} \\
&= \frac{((g_0)^{a_0} (g_1)^{b_0})^r m_0}{((g_0)^{a_0} (g_1)^{b_0})^r} \\
&= m_0
\end{aligned}$$

For $b = 1$, we have:

$$\begin{aligned}
m_1 &= c^2/(c^1)^r \\
&= \frac{h_0^{a_0} h_1^{b_0} m_0}{(g_0^{a_0} g_1^{b_0})^r} \\
&= \frac{h_0^{a_1} h_1 / g_1^{b_1} m_1}{(g_0^{a_1} g_1^{b_1})^r} \\
&= \frac{(g_0^r)^{a_1} g_1^{r+1} / g_1^{b_1} m_1}{(g_0^{a_1} g_1^{b_1})^r} \\
&= \frac{(g_0^{a_1} g_1^{b_1})^{r+1} m_1}{(g_0^{a_1} g_1^{b_1})^{r+1}} \\
&= m_1
\end{aligned}$$

Security In terms of sender security, the security can be reduced to the DDH assumption. For example, if $b = 0$, the sender sees a DDH tuple. If $b = 1$, the two values involved in the exponentiation are dependent (r and $r + 1$), but the security still reduces to the DDH assumption. In terms of receiver security, the desired message m_b can be recovered while yielding no information on m_{1-b}. We can show this by showing the following two equations are not multiples of each other:

For $k = g_0^a g_1^b$, there is a linear constraint of $\log_{g_0} k = x + y \log_{g_0} g_1$ for x and y. And for $k' = h_0^a {h'_1}^b$, we have $\log_{g_0} k' = a \log_{g_0} h_0 + b \log_{g_0} h'_1$. Because $r \neq r'$ in ciphertexts for x_{1-b}, the two equations are not multiples of each other, and thus receiver privacy is guaranteed.

From 1-2 OT to 1-4 OT (and beyond)

Either of the 1-2 OT protocols discussed above can be transformed into a more useful 1-4 OT protocol, provided we have a CPA-secure symmetric key encryption scheme [11].

Figure 3: 1-4 OT using 1-2 OT and Symmetric Encryption

Alice		**Bob**
$m_{00}, m_{01}, m_{10}, m_{11}$		$b_1 b_2$
$k_0, k_1, k'_0, k'_1 \leftarrow \{0,1\}^n$		
$k_0, k_1 \rightarrow$ 1-2 OT		$b_1 \rightarrow$ 1-2 OT
		$k_{b1} \leftarrow$ 1-2 OT
$k'_0, k'_1 \rightarrow$ 1-2 OT		$b_2 \rightarrow$ 1-2 OT
		$k'_{b2} \leftarrow$ 1-2 OT
	$c_{00} = Enc_{k_0}(Enc_{k'_0}(m_{00}))$ $\xrightarrow{\hspace{6em}}$	
	$c_{01} = Enc_{k_0}(Enc_{k'_1}(m_{01}))$ $\xrightarrow{\hspace{6em}}$	
	$c_{10} = Enc_{k_1}(Enc_{k'_0}(m_{10}))$ $\xrightarrow{\hspace{6em}}$	
	$c_{11} = Enc_{k_1}(Enc_{k'_1}(m_{11}))$ $\xrightarrow{\hspace{6em}}$	
		$m_{b_1 b_2} = Dec_{k'_{b2}}(Dec_{k_{b1}}(c_{b_1 b_2}))$

Description The above specification uses any 1-2 OT protocol to transfer one of four messages denoted $m_{00}, m_{01}, m_{10}, m_{11}$. First, Alice generates four keys, and transfers pairs of keys to Bob using 1-2 OT (note that in the above figure, a right arrow denotes sending to 1-2 OT, and a left arrow denotes receiving from 1-2 OT). In this way, Bob receives his desired keys k_{b_1} and k'_{b_2}. Alice encrypts each message using a two-layered encryption as $c_{ii'} = Enc_{k_i}(Enc_{k'i}(m_{ii'}))$. Bob is able to use his keys to decrypt his chosen ciphertext and receive his desired message.

Correctness The correctness of this protocol follows from the correctness proof for 1-2 OT, as well as from the correctness of the symmetric key system used (i.e. that $Dec_{k'_i}(Dec_{k_i}(Enc_{k_i}(Enc_{k'_i}(m_{ii'})) = m_{ii'}$ for all k_i, k'_i and $m_{ii'}$.

Security The security of this protocol follows from the security proof for 1-2 OT, as well as the security proof of CPA-security for the symmetric key system used.

Additional Notes While the construction above extends 1-2 OT to 1-4 OT, this same pattern can be utilized to achieve 1-N OT. A 1-N OT scheme using this pattern would require the generation of N keys, $\log(N)$ uses of the 1-2 OT scheme, $N\log(N)$ encryptions and $log(N)$ decryptions using the symmetric key scheme.

3 Software and Documentation

Please see the References section (at the end of the paper) for a link to the GitHub repository that contains the code used to demonstrate the various cryptographic primitives described in this paper. Please refer to the README contained in the repository, along with the following section for a description of the code.

4 Description of Software

The provided software contains Python-based implementations of 1-2 OT of the Rivest and EGL protocols discussed above. Each implementation includes both a Sender and Receiver file (the Rivest implementation also includes an Initializer file). The files are designed to be run in separate terminals in the order Receiver-Sender-Initializer or Receiver-Sender, respectively. The scripts utilize the ZeroMQ library for inter-process communication, which provides a simple way for the scripts to simulate the parties sending data over an insecure channel, such as the Internet. Each party is implemented as a class, with the main *conduct_OT()* method that sends and receives messages and performs the necessary calculations outlined in Figure 1 and 2 above. The files and README in the repository contain more detailed descriptions of each method, and also a video demonstrating the correct sequence of events to conduct oblivious transfer on simple messages. As a whole, the files provide a demonstration of the two protocols for oblivious transfer discussed in this paper. Because the main focus was simulating the protocols, there is no guarantee of security against side channel attacks. Additional code could have been written to more effectively conceal timing information, but the implementation would have taken away from the simplicity of the demonstration.

References

[1] Ronald L. Rivest. "Unconditionally Secure Commitment and Oblivious Transfer Schemes Using Private Channels and a Trusted Initializer". Massachusetts Institute of Technology. November 8, 1999.

[2] S. Even, O. Goldreich, and A. Lempel. "A Randomized Protocol for Signing Contracts". Communications of the ACM, Volume 28, Issue 6, pg. 637-647. 1985.

[3] Andrew Yao. "How to Generate and Exchange Secrets". Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pg. 162-167. October 1986.

[4] Yehuda Lindell and Benny Pinkas. "A Proof of Yao's Protocol for Secure Two-Party Computation." Journal of Cryptography, Volume 22, Issue 2, pg. 161-188. Received 21 July 2004, published 2009.

[5] Archer et. al. "From Keys to Databases - Real-World Applications of Secure Multi-Party Computation." The Computer Journal, Volume 61, Issue 12. December 2018.

[6] Michael O. Rabin. "How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University. 1981.

[7] Moni Naor and Benny Pinkas. "Oblivious transfer and polynomial evaluation." Proceedings of the 31st annual ACM Symposium on Theory of Computing, pg. 245-254. May 1999.

[8] Malkhi et. al. "Fairplay - a secure two-party computation system." Proceedings of the 13th Conference on USENIX Security Symposium, Volume 13. August 2004.

[9] Giovanni De Micheli. "Synthesis and Optimization of Digital Circuits." McGraw-Hill Higher Education. Accessed on ACM Digital Library. January 1994.

[10] Archer et. al. "From keys to databases - real-world applications for secure multi-party computation." The Computer Journal. December 2018.

[11] Katz, Jonathan. CMSC 858K - Introduction to Secure Computation. Lecture Notes. 13 September 2013.

The code used in this paper are available for use and can be found on GitHub at the following link: `https://github.com/reecetyl/CS548-Final-Paper`.