ROTed: Random Oblivious Transfer for embedded devices

P. Branco[1], L. Fiolhais[2], M. Goulão[1], P. Martins[2], P. Mateus[1] and L. Sousa[2]

[1] Instituto de Telecomunicações, Instituto Superior Técnico, Universidade de Lisboa,
{pmbranco,mgoulao,pmat}@math.tecnico.ulisboa.pt
[2] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa,
luis.azenhas.fiolhais@tecnico.ulisboa.pt, paulo.sergio@netcabo.pt, las@inesc-id.pt

Abstract. Oblivious Transfer (OT) is a fundamental primitive in cryptography, supporting protocols such as Multi-Party Computation and Private Set Intersection (PSI), that are used in applications like contact discovery, remote diagnosis and contact tracing. Due to its fundamental nature, it is utterly important that its execution is secure even if arbitrarily composed with other instances of the same, or other protocols. This property can be guaranteed by proving its security under the Universal Composability model. Herein, a 3-round Random Oblivious Transfer (ROT) protocol is proposed, which achieves high computational efficiency, in the Random Oracle Model. The security of the protocol is based on the Ring Learning With Errors assumption (for which no quantum solver is known). ROT is the basis for OT extensions and, thus, achieves wide applicability, without the overhead of compiling ROTs from OTs. Finally, the protocol is implemented in a server-class Intel processor and four application-class ARM processors, all with different architectures. The usage of vector instructions provides on average a 40% speedup. The implementation shows that our proposal is at least one order of magnitude faster than the state-of-the-art, and is suitable for a wide range of applications in embedded systems, IoT, desktop, and servers. From a memory footprint perspective, there is a small increase (16%) when compared to the state-of-the-art. This increase is marginal and should not prevent the usage of the proposed protocol in a multitude of devices. In sum, the proposal achieves up to 37k ROTs/s in an Intel server-class processor and up to 5k ROTs/s in an ARM application-class processor. A PSI application, using the proposed ROT, is up to 6.6 times faster than related art.

Keywords: Oblivious Transfer, Embedded Systems, Private Set Intersection, Universal Composability, Post-Quantum Cryptography

1 Introduction

Oblivious Transfer (OT) is one of the most fundamental primitives in cryptography. A typical OT protocol involves two parties: a sender, which inputs two messages (m_0, m_1), and a receiver, which inputs a bit $b \in \{0, 1\}$. At the end, the receiver outputs m_b. In terms of security, it is required that the sender learns nothing about the bit b and that the receiver learns nothing about m_{1-b}. In this work, we study a variant of OT, which is called Random OT (ROT). In this variant, neither the sender nor the receiver have any inputs. Instead, the protocol should output (m_0, m_1) to the sender and (b, m_b) to the receiver, where (m_0, m_1) are messages chosen uniformly at random and b is a uniform bit.

It is well-known that OT is complete for secure Two-Party Computation (2PC) [Yao82] and Multiparty Computation (MPC) [GMW87]. However, the number of OTs needed to implement these protocols scales with the size of the circuit, which turns their efficiency completely prohibitive for real-world applications. To overcome this problem, Ishai *et al.* [IKNP03] showed how to efficiently extend a small number of *base* OTs into a large number of OT correlations using only cheap symmetric-key operations. It turns out that, to use these extension techniques, we need to use ROT instances as base OTs in the malicious setting [OOS17]. Since compiling ROTs from OTs will introduce a significant time overhead, it is crucial to design efficient ROT directly from basic hardness assumptions.

While ROTs are the basis for the implementation of OT extensions, most related art has focused on the design of standard OTs [PVW08, CO15, HL17]. Also, the most efficient implementations of base OTs are based on number theoretic assumptions [CO15, HL17]. The implementation of OT protocols from these assumptions has led to practical solutions for this problem. While these could be adapted for the ROT setting, number theoretic assumptions are known to be insecure in the presence of quantum adversaries. This raises the question of whether we are still able to efficiently and securely realize 2PC and MPC in a post-quantum world.

The main goal of this article is to design and implement an efficient ROT protocol, based on security assumptions that are thought to be hard to break even in a post-quantum setting. By focusing on the ROT setting instead of the standard OT definition, we design a protocol with a reduced number of exchanged messages and better computational efficiency. Moreover, the protocol can be plugged directly in OT extensions without having the time overhead of compiling ROTs from OTs, which is required in the malicious setting.

We demonstrate the improvements and applicability of the proposed protocol by implementing it in multiple processor architectures, and benchmarking them against the current state-of-the-art. Finally, we integrate the ROT protocol as part of a state-of-the-art Private Set Intersection (PSI) protocol.

Applications of ROT OT is a ubiquitous primitive in cryptography. Its applications range from 2PC/MPC to zero-knowledge proof systems [PSSW09, HOSS18, KMO90]. Recently, a long line of works has used ROT to design efficient PSI protocols in a variety of settings [PRTY19, PRTY20a]. PSI is a protocol executed between two parties: each party inputs a list of elements, and receives as output a list of the elements that were simultaneously present in the lists of both parties, without learning anything else about the other party's list. Designing a quantum-safe PSI protocol is fundamental to future-proof its many applications, which include:

Contact discovery [DRRT18]: Most social-networking applications involve an initialization step, in which the list of contacts where the application is installed is compared with a list of users in a centralised server, to identify which of the user's contacts already use the service. PSI may be used to improve both the privacy of the new user, as well as of the users of the social-networking application.
Remote diagnosis [BPSW07]: A system may gather information about a problem and send it to a centralised server for diagnosis. The information gathered by the system may be sensitive. When a computer system is considered, this information might include passwords or company-owned data; in the medical industry, this might include a patient's health records. The company running the diagnosis might not want to risk disclosing its proprietary diagnostic programs. PSI may be used in this settings to guarantee the privacy of both parties.
Contact tracing [ABC$^+$20]: Two devices may track their users' location. At a certain point, the two users may wish to know if they have been in the same place, but do not wish to disclose anything more about where they have been. This use-case may be useful to trace the contacts of a COVID-infected person. PSI may be used to protect the privacy of the involved parties.
Among others [ISMG20, IKN$^+$19].

These applications may have different computational requirements. On the one hand, contact discovery and contact tracing are typically executed on smartphones with limited computational power and power constraints. In this case, it is important to minimize the number of exchanged messages and consumed memory. On the other hand, remote diagnoses may be performed by high-performance computing platforms on very large sets. In this case, latency should be minimized.

1.1 Contributions

ROT protocol from RLWE We start by designing a ROT protocol from the Ring Learning with Errors (RLWE) assumption [LPR10, LPR13] in the Random Oracle Model (ROM). The protocol runs in three rounds and we prove its security in the Universal Composability (UC) model [Can01] in the presence of malicious adversaries.

Our new protocol is inspired by the recent work of [BDGM19], which presents an OT protocol from the hardness of the RLWE assumption that runs in four rounds. Since our goal is to design a ROT protocol, we can actually reduce the number of rounds of the protocol to three while improving the efficiency of the protocol. In contrast, adapting [BDGM19] to a ROT with a black-box approach would introduce three further messages (cf. Section 4). Having a lower number of exchanged messages significantly reduces the communication latency and consumed energy, which is of particular importance for applications executing on constrained computing platforms.

ROT Protocol Implementation The ROT protocol was implemented in C++ with state-of-the-art libraries in order to achieve the best performance [Qua, BT]. We provide an analysis of the bottlenecks in the implementation as well as insights on how these were improved. The computational requirements, the performance, and the memory consumption of the new protocol are experimentally evaluated in an Intel server-class processor and in three ARM application-class processors. The results show that our proposal is at least one order of magnitude faster than the state-of-the-art, and is suitable for a wide range of applications in embedded systems, IoT, desktop, and servers.

Practical PSI use-case In order to understand the impact of the proposed protocol in a real-world application, we integrated the proposed ROT in an open-source PSI framework [PRTY20b]. Similarly to the protocol implementation, we provide an analysis for the server-class processor encompassing memory requirements and performance.

2 Preliminaries

As usual, $\mathbb{N}$ denotes the set of natural numbers, $\mathbb{Z}$ denotes the set of integers, $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$, for any $q \in \mathbb{N}$, and $\mathbb{Z}[X]$ (resp. $\mathbb{Z}_q[X]$) denotes the ring of polynomials on variable X with coefficients in $\mathbb{Z}$ (resp. $\mathbb{Z}_q$). If $\mathcal{A}$ is an algorithm, we denote by $y \leftarrow \mathcal{A}(x)$ the output of the experiment of running $\mathcal{A}$ on input x. If S is a set, we denote by $x \leftarrow\!\!\$\, S$ the experiment of choosing uniformly at random an element x from S. If χ is a probabilistic distribution over some set S, $x \leftarrow\!\!\$\, \chi$ denotes the experiment of sampling an element x from S according to χ. If x and y are two binary strings, we denote by $x|y$ their concatenation and by $x \oplus y$ their bit-wise XOR. If X and Y are two probability distributions, $X \approx Y$ means that they are computationally indistinguishable. A negligible function $\mathsf{negl}(n)$ is a function such that $\mathsf{negl}(n) < 1/\mathsf{poly}(n)$ for every polynomial $\mathsf{poly}(n)$ and sufficiently large n. By a PPT algorithm we mean a probabilistic polynomial-time algorithm.

2.1 UC security and ideal functionalities

The UC framework, introduced by Canetti [Can01], ensures that the security of a protocol does not depend on other executions of the same or other protocols. Let π be a protocol where n parties and an adversary $\mathcal{A}$ are involved. We denote the output of the environment $\mathcal{E}$ at the end of the real-world execution of π with adversary $\mathcal{A}$ by $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$. The output of $\mathcal{E}$ at the end of the ideal-world execution of a functionality $\mathcal{F}$ with adversary Sim is denoted by $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{E}}$. The following definition introduces the notion of a protocol emulating (in a secure way) some ideal functionality.

Definition 1. We say that a protocol π *UC-realizes* $\mathcal{F}$ if, for every PPT adversary $\mathcal{A}$, there is a PPT simulator Sim, such that for all PPT environments $\mathcal{E}$, $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{E}} \approx \mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{E}}$, where $\mathcal{F}$ is an ideal functionality.

In this work, we consider *static malicious* adversaries which are adversaries that may deviate in any way from the protocol, but the corruption of each party happens before the beginning of the protocol.

Random oracle ideal functionality We work in the so-called $\mathcal{F}_{\mathrm{RO}}$-hybrid model in order to model random oracles in the UC framework.[1] The random oracle ideal functionality $\mathcal{F}_{\mathrm{RO}}$ is presented below. Let $\mathcal{D}$ be the range of the random oracle and L be a list, which is initially empty. The value sid represents the session ID and the parties involved in the protocol. Notwithstanding, we will often not explicitly specify sid (but sid is implicit) as argument of a query (*e.g.* write $\mathsf{H}(q)$ meaning $\mathsf{H}(\mathsf{sid}|q)$) to avoid clutter in the notation.

$\mathcal{F}_{\mathbf{RO}}$ functionality

Upon receiving a query $(\mathsf{sid}|q)$ from a party P or from an adversary $\mathcal{A}$, $\mathcal{F}_{\mathrm{RO}}$ proceeds as follows:

- If there is a pair $(q, h) \in L$ it returns $(\mathsf{sid}|h)$;
- Else, it chooses $h \leftarrow_\$ \mathcal{D}$, stores the pair $(q, h) \in L$ and returns $(\mathsf{sid}|h)$.

ROT ideal functionality We now present the ideal functionality for ROT.

$\mathcal{F}_{\mathbf{ROT}}$ functionality

• Upon receiving a message $(\mathsf{sid}, \mathsf{start})$ from both R and S, $\mathcal{F}_{\mathrm{ROT}}$ samples $M_0, M_1 \leftarrow_\$ \{0,1\}^\kappa$ (where κ is the security parameter) and $b \leftarrow_\$ \{0,1\}$. It sends (sid, M_0, M_1) to S and (sid, b, M_b) to R.

2.2 Ring Learning With Errors

The RLWE problem [LPR10] is the ring version of the Learning with Errors (LWE) problem [Reg05] and it is conjectured to be hard for both classical and quantum computers. Before presenting the problem, we define the RLWE distribution. Let $q \geq 2$, $R_q = \mathbb{Z}_q[X]/\langle f(X)\rangle$ where $f(X)$ is the nth-cyclotomic polynomial, and χ be the error distribution (which is usually a discrete Gaussian [LPR10]) and which satisfies $\Pr[\|p\| > \beta : p \leftarrow_\$ \chi] \leq \mathsf{negl}(n)$ for some $\beta \in \mathbb{N}$, where $\|p\| = \|p\|_\infty = \max_i\{p_i\}$ denotes the largest coefficient of the polynomial $p = p_0 + p_1X + \dots p_{n-1}X^{n-1} \in R_q$. For $s \in R_q$, the RLWE distribution $A_{s,\chi}$ is obtained by choosing $a \leftarrow_\$ R_q$, $e \leftarrow_\$ \chi$, and outputting $(a, as + e \mod q)$.

[1] Recall that UC-secure OT is impossible in the plain model [CF01].

Definition 2 (Ring Learning with Errors)**.** *Let n, q, R_q, χ and $A_{s,\chi}$ be as above. The decision version of the RLWE problem is the following: for $s \leftarrow_\$ R_q$, distinguish the case when it is given a polynomial number of samples from $A_{s,\chi}$ or when it is given uniformly chosen at random values from $R_q \times R_q$.*

The RLWE problem is proven to be as hard as quantumly solving a worst-case lattice problem (the approximate Shortest Vector Problem (SVP) on ideal lattices) which is considered to be hard for both classical and quantum computers [LPR10]. The main advantages of using the RLWE instead of the LWE assumption are the smaller key-size and the ability to use the Number Theoretic Transform (NTT) to enhance the speed of the operations. Here, we use the Hermite Normal Form of the RLWE problem, usually called HNF-RLWE, in which the secret s is sampled from the error distribution χ, instead of being chosen uniformly at random from the ring R_q. The HNF-RLWE reduces to RLWE, and so this version of the problem is also assumed to be hard [ACPS09].

Now, we address the reconciliation mechanisms of the Key Exchange (KE) of [DXL12]. We define the signal function Sig and the extraction function Mod_2 as in [DXL12]. Both these functions are used in the reconciliation mechanism of the KE protocol and allow the involved parties to compute a shared key. Let $\sigma_0, \sigma_1 : \mathbb{Z}_q \to \{0,1\}$ such that

$$\sigma_0(a) = \begin{cases} 0, & a \in \left[-\lfloor \frac{q}{4} \rfloor, \lfloor \frac{q}{4} \rfloor\right] \\ 1, & \text{otherwise} \end{cases} \qquad \text{and} \quad \sigma_1(a) = \begin{cases} 0, & a \in \left[-\lfloor \frac{q}{4} + 1 \rfloor, \lfloor \frac{q}{4} + 1 \rfloor\right] \\ 1, & \text{otherwise} \end{cases}$$

for $a \in \mathbb{Z}_q$. When $a = \sum_{i=0}^{n-1} a_i X^i \in R_q$, then $\sigma_0(a) = \sum_{i=0}^{n-1} \sigma_0(a_i) X^i$ and $\sigma_1(a) = \sum_{i=0}^{n-1} \sigma_1(a_i) X^i$. The signal function $\mathsf{Sig} : R_q \to R_2$ is defined as $\mathsf{Sig}(a) = \sigma_b(a)$ where $b \leftarrow_\$ \{0,1\}$. The extraction function $\mathsf{Mod}_2 : R_q \times R_2 \to R_2$ is defined as

$$\mathsf{Mod}_2(a, \sigma) = \left(a + \sigma \frac{q-1}{2} \mod q\right) \mod 2.$$

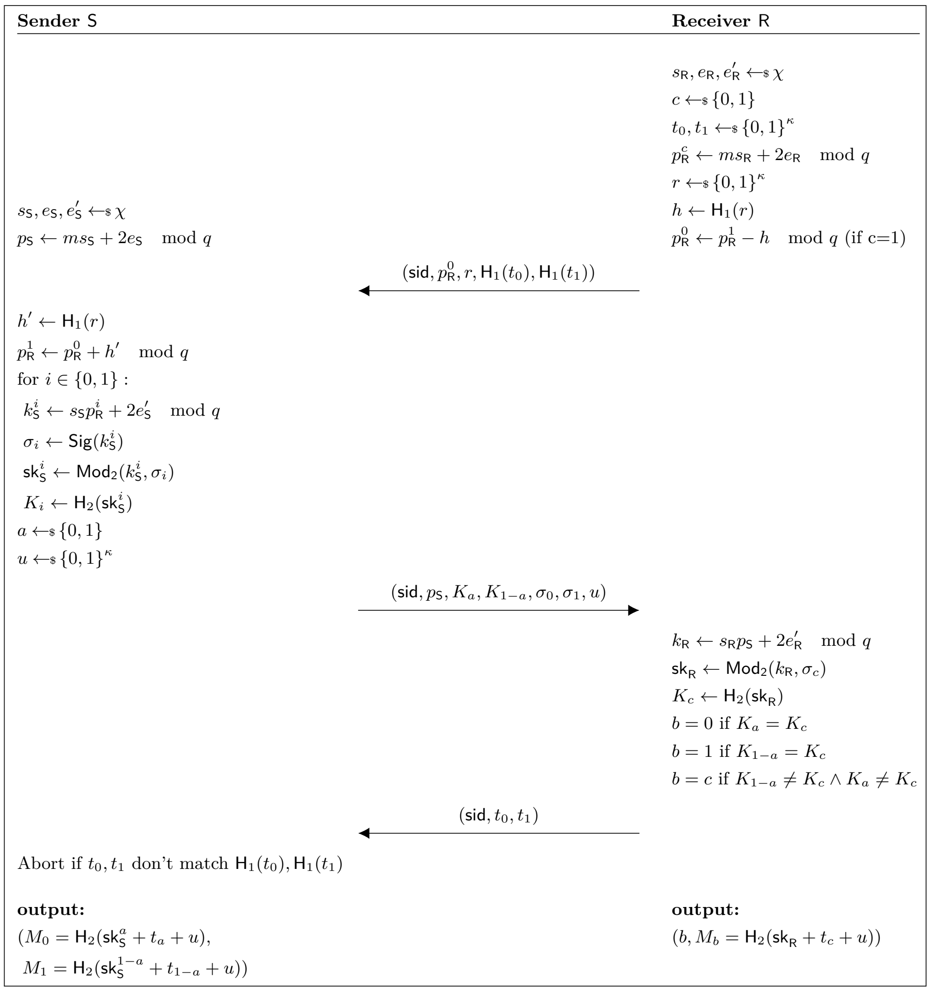
3 ROT protocol from RLWE

In this section, we present our ROT protocol which can be seen as a tweaked version of the protocol of [BDGM19] (1-out-of-2 OT), albeit with improved round complexity and without requiring a symmetric encryption scheme. Then, we show that the protocol is UC-secure under the RLWE assumption in the ROM.

It is well-known that it is impossible to achieve (maliciously) UC-secure OT in the plain model [CF01]. Hence, we use the ROM in our security proofs. We note that our security proof holds on the hardness of RLWE, which is believed to be secure against quantum adversaries. However, UC security using the ROM does not consider an adversary that can query the random oracle in *superposition* – usually called the Quantum Random Oracle Model (QROM).[2] Since our application scenarios are OT extensions and PSI, both also not proven secure for QROM, we leave as an open problem how to extend our proof.

Intuitively, the protocol works by partially running two KEs in parallel. First, the receiver samples one authentic KE message for which it knows the secrets, and a fake (but indistinguishable) one which it is forced to make uniformly random using the ROM. Second, the sender samples its secrets and KE message, and runs the two reconciliations, resulting in two keys which it then uses to hide the two ROT messages. Third, the receiver runs the reconciliation for the authentic exchange and recovers one of the ROT messages.

[2] While there are examples of schemes that are secure in the ROM but insecure in the QROM [BDF+11], we stress that such schemes are specially crafted so that a quantum adversary can attack them.



```
Sender S                                                      Receiver R

                                                              $s_R, e_R, e'_R \leftarrow_{\$} \chi$
                                                              $c \leftarrow_{\$} \{0,1\}$
                                                              $t_0, t_1 \leftarrow_{\$} \{0,1\}^\kappa$
                                                              $p_R^c \leftarrow m s_R + 2e_R \mod q$
                                                              $r \leftarrow_{\$} \{0,1\}^\kappa$
$s_S, e_S, e'_S \leftarrow_{\$} \chi$                         $h \leftarrow H_1(r)$
$p_S \leftarrow m s_S + 2e_S \mod q$                          $p_R^0 \leftarrow p_R^1 - h \mod q$ (if c=1)

                 $\xleftarrow{(sid, p_R^0, r, H_1(t_0), H_1(t_1))}$

$h' \leftarrow H_1(r)$
$p_R^1 \leftarrow p_R^0 + h' \mod q$
for $i \in \{0,1\}$ :
  $k_S^i \leftarrow s_S p_R^i + 2e'_S \mod q$
  $\sigma_i \leftarrow Sig(k_S^i)$
  $sk_S^i \leftarrow Mod_2(k_S^i, \sigma_i)$
  $K_i \leftarrow H_2(sk_S^i)$
$a \leftarrow_{\$} \{0,1\}$
$u \leftarrow_{\$} \{0,1\}^\kappa$

                 $\xrightarrow{(sid, p_S, K_a, K_{1-a}, \sigma_0, \sigma_1, u)}$

                                                              $k_R \leftarrow s_R p_S + 2e'_R \mod q$
                                                              $sk_R \leftarrow Mod_2(k_R, \sigma_c)$
                                                              $K_c \leftarrow H_2(sk_R)$
                                                              $b = 0$ if $K_a = K_c$
                                                              $b = 1$ if $K_{1-a} = K_c$
                                                              $b = c$ if $K_{1-a} \neq K_c \wedge K_a \neq K_c$

                 $\xleftarrow{(sid, t_0, t_1)}$

Abort if $t_0, t_1$ don't match $H_1(t_0), H_1(t_1)$

output:                                                       output:
$(M_0 = H_2(sk_S^a + t_a + u),$                               $(b, M_b = H_2(sk_R + t_c + u))$
 $M_1 = H_2(sk_S^{1-a} + t_{1-a} + u))$
```

Figure 1: 3-round ROT protocol based on the RLWE assumption.

In the following setting, let χ and q be as in Definition 2 and κ be the security parameter. Let $\mathsf{H}_1 : \{0,1\}^\kappa \to R_q$, $\mathsf{H}_2 : R_2 \to \{0,1\}^\kappa$ be hash functions modeled as Random Oracles (ROs). Let $m \leftarrow_{\$} R_q$ be a public uniformly chosen ring element which can be obtained by querying a RO in some predefined input. Let Mod_2 and Sig be the algorithms defined in Section 2.2. The protocol is presented in Fig. 1, where the parties (sender S and receiver R) have no input written on their input tape. Then, S outputs two uniform random messages (M_0, M_1), and R outputs a uniform random bit and the corresponding message (b, M_b).

Theorem 1. *The protocol presented in Figure 1 is correct.*

Proof. Let (M_0, M_1) be the output of the sender and (b, M') be the output of the receiver. To prove that the protocol is correct we have to show that $M_b = M'$. By the correctness of [DXL12], $\mathsf{sk}_\mathsf{S}^c = \mathsf{sk}_\mathsf{R}$ except with negligible probability.

Now if $a = c$, then $\mathsf{sk}_\mathsf{S}^a = \mathsf{sk}_\mathsf{R}$ and thus $K_a = K_c$. In this case, b is set to 0, $M_0 = \mathsf{H}_2(\mathsf{sk}_\mathsf{S}^a + t_a + u)$ and $M' = \mathsf{H}_2(\mathsf{sk}_\mathsf{R} + t_c + u)$. So, we conclude that $M_b = M'$.

Analogously, if $a \neq c$ then $1 - a = c$ and $\mathsf{sk}_\mathsf{S}^{1-a} = \mathsf{sk}_\mathsf{R}$ and thus $K_{1-a} = K_c$. In this case, b is set to 1, $M_1 = \mathsf{H}_2(\mathsf{sk}_\mathsf{S}^{1-a} + t_{1-a} + u)$ and $M' = \mathsf{H}_2(\mathsf{sk}_\mathsf{R} + t_c + u)$. Again, we conclude that $M_b = M'$. □

Intuitively, the computational security of the protocol can be derived as follows.

A corrupt sender cannot learn the bit b, because while it holds two KE messages from the receiver (p_{R}^0 and p_{R}^1), only one of these was generated as an RLWE sample (p_{R}^c). The other message (p_{R}^{1-c}) is coerced to be a uniform random element by summing or subtracting a random value obtained from the RO, and distinguishing the two yields the bit b but means breaking the RLWE assumption.

A corrupt receiver cannot learn both messages, as computing each message requires computing a shared key with the sender for p_{R}^0 and p_{R}^1. Again, only p_{R}^c was generated as an RLWE sample, while p_{R}^{1-c} is uniformly random. Therefore, only the shared key corresponding to p_{R}^c may be computed and output, otherwise the adversary needs to break the RLWE assumption and find the secrets for p_{R}^{1-c}.

To prove UC security against one of the parties, we need to build a simulator that is able to program the output of the protocol while not being noticed by the adversary.

For a corrupted sender, the simulator can program H_1 in such a way that it is able to recover both keys $\mathsf{sk}_{\mathsf{S}}^i$ obtained by the sender (this can be done by programming H_1 to output an RLWE sample, which is indistinguishable from uniform output). Since the simulator now has both keys, it can extract the value a from the malicious sender (in case a is not fully specified by the transcript, then the simulator sets $a \leftarrow_\$ \{0,1\}$ which goes unnoticed to the adversary because of the third condition, $b = c$ *if* $K_{1-a} \neq K_c \wedge K_a \neq K_c$, in the real protocol). In the end, it can program H_2 to output the right messages it received from the ideal functionality (M_0, M_1).

For a corrupted receiver, to enforce the adversary to output the random message M, the simulator programs the oracle H_2 to output M when queried on the correct input. And, to program the random bit b output by the receiver, the sender simply needs to extract the bit c sampled by the receiver. So, to extract the bit c from the receiver, the simulator checks if the receiver queries H_1 on $\mathsf{sk}_{\mathsf{S}}^0$ or $\mathsf{sk}_{\mathsf{S}}^1$. When this happens, the simulator can program the RO to set the bit a in such a way that it specifies b to be the same bit output by the ideal functionality.

Theorem 2. *The protocol UC-realizes the $\mathcal{F}_{ROT}$ functionality against static malicious adversaries, given that the HNF-RLWE assumption holds.*

Proof. In the proof of UC security, we analyze the four possible cases for the execution of the ROT two-party protocol with an adversary present, by describing the step-by-step procedure of the simulator. The simulator runs the adversary as a black-box, such that the execution in the ideal-world mimics the real-world execution to the view of the external environment, in order to attain security.

Security against a corrupted sender. We first describe the simulator Sim for a corrupted sender. Let $\mathcal{A}$ be the adversary corrupting the sender.

1. Sim starts by receiving (M_0, M_1) from $\mathcal{F}_{\mathrm{ROT}}$. It answers queries to the ROs as an $\mathcal{F}_{\mathrm{RO}}$ would, unless explicitly specified otherwise.

2. It sets $c \leftarrow_\$ \{0,1\}$, $t_0, t_1 \leftarrow_\$ \{0,1\}^\kappa$. It computes $p_{\mathsf{R}}^0 \leftarrow m s_{\mathsf{R}}^0 + 2e_{\mathsf{R}}^0$ and $p_{\mathsf{R}}^1 \leftarrow m s_{\mathsf{R}}^1 + 2e_{\mathsf{R}}^1$, two RLWE samples. It chooses $r \leftarrow_\$ \{0,1\}^\kappa$ and programs H_1 such that $\mathsf{H}_1(r) = p_{\mathsf{R}}^1 - p_{\mathsf{R}}^0$. It sends $(\mathsf{sid}, p_{\mathsf{R}}^0, r, \mathsf{H}_1(t_0), \mathsf{H}_1(t_1))$ to $\mathcal{A}$.

3. Upon receiving $(\mathsf{sid}, p_{\mathsf{S}}, K_a, K_{1-a}, \sigma_0, \sigma_1, u)$. It recovers the shared keys by computing $\mathsf{sk}_{\mathsf{R}}^i \leftarrow \mathsf{Mod}_2(s_{\mathsf{R}}^i p_{\mathsf{S}} + 2{e'}_{\mathsf{R}}^i, \sigma^i)$ for $i \in \{0,1\}$ and extracts a. To extract a, the Sim checks if $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^0) = K_a$ and $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^1) = K_{1-a}$ (in which case $a = 0$), or if $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^1) = K_a$ and $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^0) = K_{1-a}$ (in which case $a = 1$). Else, it sets $a \leftarrow_\$ \{0,1\}$.

4. The simulator sends (sid, t_0, t_1) as the honest receiver would. Finally, the simulator programs H_2 to output M_0 on input $\mathsf{sk}_{\mathsf{S}}^a + t_a + u$ and M_1 on input $\mathsf{sk}_{\mathsf{S}}^{1-a} + t_{1-a} + u$, since it has $\mathsf{sk}_{\mathsf{S}}^0, \mathsf{sk}_{\mathsf{S}}^1, a$.

We now argue that the real execution of the protocol is indistinguishable from simulated one. The proof follows from the indistinguishability of the following hybrid distributions.

Hybrid $\mathcal{H}_0$. The real-world execution of the protocol. In particular, in this hybrid, the simulator behaves exactly as the honest receiver would do.

Hybrid $\mathcal{H}_1$. Identical to $\mathcal{H}_0$, except that the simulator aborts if the adversary had already queried H_1 on r. Then, it programs H_1 to output $p_{\mathsf{R}}^1 - p_{\mathsf{R}}^0$ on r, with $p_{\mathsf{R}}^0 \leftarrow ms_{\mathsf{R}}^0 + 2e_{\mathsf{R}}^0$ and $p_{\mathsf{R}}^1 \leftarrow ms_{\mathsf{R}}^1 + 2e_{\mathsf{R}}^1$ (as it is described in Step 2).

Hybrid $\mathcal{H}_2$. Identical to $\mathcal{H}_1$, except that the simulator first extracts a (as described in step 3), and then programs H_2 to output M_0 on input $\mathsf{sk}_{\mathsf{S}}^a + t_a + u$, and M_1 on input $\mathsf{sk}_{\mathsf{S}}^{1-a} + t_{1-a} + u$.

Claim 1. *Hybrids $\mathcal{H}_0$, $\mathcal{H}_1$ are indistinguishable given that the RLWE assumption holds.*

Proof. First, since $r \leftarrow_{\$} \{0,1\}^\kappa$, the probability that the simulator aborts as the adversary queries H_1 on r, before seeing r, is exponentially small in κ.

Then, the differences lie only in the values p_{R}^0 and $p_{\mathsf{R}}^1 \leftarrow p_{\mathsf{R}}^0 + \mathsf{H}_1(r)$. In $\mathcal{H}_0$, p_{R}^c is an RLWE sample and p_{R}^{1-c} is a uniform random sample, given c. In the hybrid $\mathcal{H}_1$, $p_{\mathsf{R}}^0, p_{\mathsf{R}}^1$ are both RLWE samples. Clearly, distinguishing both hybrids is breaking the RLWE assumption, by distinguishing an RLWE sample from a uniform random value. $\square$

Claim 2. *Hybrids $\mathcal{H}_1$, $\mathcal{H}_2$ are indistinguishable.*

Proof. First, M_0, M_1, output in $\mathcal{H}_2$, are uniform random values, which come from $\mathcal{F}_{\mathrm{ROT}}$, and, as such, they are indistinguishable from ideal outputs of $\mathcal{F}_{\mathrm{RO}}$, output in $\mathcal{H}_1$. All other queries are the same in both executions as they are answered by the simulator like an ideal functionality $\mathcal{F}_{\mathrm{RO}}$.

Then, the case where the simulator is not able to extract a happens only when $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^0) \neq K_a$ or $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^1) \neq K_{1-a}$, and when $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^1) \neq K_a$ or $\mathsf{H}_2(\mathsf{sk}_{\mathsf{R}}^0) \neq K_{1-a}$. In this case, the simulator sets $a \leftarrow_{\$} \{0,1\}$. For this, the third condition stated in the real protocol, $b = c$ *if* $K_{1-a} \neq K_c \wedge K_a \neq K_c$, guarantees that no information about a is leaked to the adversary. Hence, the executions remain indistinguishable for both worlds. $\square$

Finally, note that hybrid $\mathcal{H}_2$ describes the simulated protocol. So, the simulator successfully simulates the real-world adversary and the execution trace is indistinguishable from a real-world execution $\mathcal{H}_0$, assuming the hardness of RLWE, and except with negligible probability in κ. This concludes the proof of security against a corrupted sender.

Security against a corrupted receiver. Then, we describe the simulator Sim for a corrupted receiver. Let $\mathcal{A}$ be the adversary corrupting the receiver.

1. Sim first receives (b, M) from $\mathcal{F}_{\mathrm{ROT}}$. It answers queries to the ROs as an $\mathcal{F}_{\mathrm{RO}}$ would, unless explicitly specified otherwise.

2. It waits until receiving the first message $(\mathsf{sid}, p_{\mathsf{R}}^0, r, \mathsf{H}_1(t_0), \mathsf{H}_1(t_1))$, and samples $a \leftarrow_{\$} \{0,1\}$, $u \leftarrow_{\$} \{0,1\}^\kappa$, $K_i \leftarrow_{\$} \{0,1\}^\kappa$ and computes $p_{\mathsf{S}}, \mathsf{sk}_{\mathsf{S}}^i, \sigma_i$ honestly (for $i \in \{0,1\}$). It sends $(\mathsf{sid}, p_{\mathsf{S}}, K_a, K_{1-a}, \sigma_0, \sigma_1, u)$ to $\mathcal{A}$.

3. It programs H_2 when queried on $\mathsf{sk}_{\mathsf{S}}^0$ or $\mathsf{sk}_{\mathsf{S}}^1$ (if queried on both, aborts) to answer K_a when $b = 0$, and K_{1-a} otherwise. And, it programs H_2 when queried on $\mathsf{sk}_{\mathsf{S}}^0 + t_0 + u$ or $\mathsf{sk}_{\mathsf{S}}^1 + t_1 + u$ (if queried on both, aborts) to output M.[3]

[3] Sim knows t_0, t_1 from observing H_1. If the values were not observed, Sim answers H_2 as an ideal RO.

4. As in the honest protocol, it checks the hashes of t_0, t_1 and aborts if they do not match the ones sent in the first message of the protocol.

Again, we argue the indistinguishability of the real and simulated traces by showing the sequential indistinguishability of the hybrid distributions defined by these executions.

Hybrid $\mathcal{H}_3$. The real-world execution of the protocol. In particular, in this hybrid, the simulator behaves exactly as the honest sender would do.

Hybrid $\mathcal{H}_4$. Identical to $\mathcal{H}_3$, but programming H_2 when queried on sk_S^0 or sk_S^1 to answer with K_a when $b = 0$, and K_{1-a} when $b = 1$. Aborting if both sk_S^0 and sk_S^1 are queried.

Hybrid $\mathcal{H}_5$. Identical to $\mathcal{H}_4$, but programming H_2 when queried on $\mathsf{sk}_\mathsf{S}^0 + t_0 + u$ or $\mathsf{sk}_\mathsf{S}^1 + t_1 + u$ to output M. Aborting if both $\mathsf{sk}_\mathsf{S}^0 + t_0 + u$ and $\mathsf{sk}_\mathsf{S}^1 + t_1 + u$ are queried.

Claim 3. *Hybrids $\mathcal{H}_3$, $\mathcal{H}_4$ are indistinguishable given that the RLWE assumption holds.*

Proof. The oracle H_2 is programmed to reply to sk_S^0 or sk_S^1 with K_a or K_{1-a}, depending on the input b from $\mathcal{F}_{\mathrm{ROT}}$, in order to force this b to be output by $\mathcal{A}$. Since $\mathcal{A}$ does not know any information about b (it comes from $\mathcal{F}_{\mathrm{ROT}}$), this is indistinguishable.

Then, there is the possibility of the simulator aborting if both sk_S^0 and sk_S^1 are queried on the oracle. If $\mathcal{A}$ could get any information about $\mathsf{sk}_\mathsf{S}^{1-c}$ from only public information of the KE ($p_\mathsf{S}, \sigma_0, \sigma_1$), then it would break its security [DXL12] since it provides a distinguisher for the KE shared keys. So, $\mathcal{A}$ knowing both $\mathsf{sk}_\mathsf{R} = \mathsf{sk}_\mathsf{S}^c$ and $\mathsf{sk}_\mathsf{S}^{1-c}$ would mean that it knew both k_R^c $(= k_\mathsf{R} = s_\mathsf{R} p_\mathsf{S} + 2e'_\mathsf{R})$ from which it reconciles sk_R, and k_R^{1-c} $(= s_\mathsf{R}^* p_\mathsf{S} + 2e_\mathsf{R}^*)$ allowing it to reconcile also $\mathsf{sk}_\mathsf{S}^{1-c}$. While $\mathcal{A}$ may compute k_R^c from $s_\mathsf{R}, e'_\mathsf{R}$ (acting honestly), to know k_R^{1-c} it would need to find $s_\mathsf{R}^*, e_\mathsf{R}^*$ such that $p_\mathsf{R}^{1-c} = m s_\mathsf{R}^* + 2e_\mathsf{R}^*$. However, p_R^{1-c} is uniformly random (due to summing or subtracting h), and so, computing $s_\mathsf{R}^*, e_\mathsf{R}^*$ (which may not even exist) is equivalent to breaking the RLWE assumption, thus the execution is indistinguishable from $\mathcal{H}_3$, up to a negligible probability in κ.

Mind that the case where $K_{1-a} \neq K_c \wedge K_a \neq K_c$ never happens when the sender is honest, thus the simulator will always answer K_a or K_{1-a} when queried on sk_R, independently of it being sk_S^0 or sk_S^1. If none is asked, then the execution is also indistinguishable. □

Claim 4. *Hybrids $\mathcal{H}_4$, $\mathcal{H}_5$ are indistinguishable given that the RLWE assumption holds.*

Proof. First, programming the oracle H_2 when queried on $\mathsf{sk}_\mathsf{S}^0 + t_0 + u$ or $\mathsf{sk}_\mathsf{S}^1 + t_1 + u$ to output M is indistinguishable from the execution of $\mathcal{H}_4$. Since the reply M (which comes from $\mathcal{F}_{\mathrm{ROT}}$) from $\mathcal{H}_5$ is a random string of appropriate length, it is indistinguishable from some uniform random value output by the RO from $\mathcal{H}_4$.

Then there is the case that the simulator aborts if both $\mathsf{sk}_\mathsf{S}^0 + t_0 + u$ and $\mathsf{sk}_\mathsf{S}^1 + t_1 + u$ are queried to the oracle H_2. As in Claim 3, $\mathcal{A}$ cannot know both sk_S^0 and sk_S^1, given that the RLWE assumption holds, up to negligible probability in κ. So, the case that here the simulator aborts without the real-world adversary also aborting is also negligible in κ. □

Certainly, $\mathcal{H}_3$ and $\mathcal{H}_5$ are indistinguishable, up to negligible probability in κ, given the hardness of the RLWE assumption. And, since $\mathcal{H}_3$ represents the real-world execution of the protocol, and $\mathcal{H}_5$ represents the corresponding ideal-world simulation, this ends our proof for the security against a corrupted receiver.

Security for the remaining cases. To conclude, we describe the simulation for the case, when neither the sender nor the receiver are corrupted by the adversary, and when both the sender and receiver are corrupted.

When no party is corrupted (*i.e.* the adversary is not corrupting any party), the simulator has no input from the ideal functionality $\mathcal{F}_{\mathrm{ROT}}$, as the adversary being simulated is not actually playing a party in the real-world protocol. So, the simulator generates and honestly executes the protocol for dummy outputs (M_0, M_1) for the sender and (b, M_b) for the receiver, where $b \leftarrow_{\$} \{0,1\}$ and $M_0, M_1 \leftarrow_{\$} \{0,1\}^{\kappa}$, and forwards the messages of honestly simulated protocol to the adversary (again, which just observes the transcript). Now, the transcript has three messages, which the simulator generates, and which must be shown to be indistinguishable from a real-world execution of the protocol.

1. $(\mathsf{sid}, p_{\mathsf{R}}^0, r, \mathsf{H}_1(t_0), \mathsf{H}_1(t_1))$, has three uniform random values $(r, \mathsf{H}_1(t_0), \mathsf{H}_1(t_1))$, which are statistically indistinguishable from any dummy uniform random values that the simulator generates. As for p_{R}^0, it is either an RLWE sample or a uniform random value (from which p_{R}^1 could be computed), and, from the hardness of the RLWE assumption, it is indistinguishable from any dummy RLWE sample or dummy uniform random value generated by the simulator.

2. $(\mathsf{sid}, p_{\mathsf{S}}, K_a, K_{1-a}, \sigma_0, \sigma_1, u)$, has three random values (K_a, K_{1-a}, u) which are statistically indistinguishable from dummy uniform random values from the simulation. Regarding $(p_{\mathsf{S}}, \sigma_0, \sigma_1)$, from the hardness of the RLWE [DXL12], these do not leak information, since only p_{R}^0 or p_{R}^1 may be known (the adversary cannot even know whether it knows p_{R}^0 or p_{R}^1), and so, these values are indistinguishable from the simulated ones.

3. (sid, t_0, t_1) has two uniform random values t_0, t_1, such that $\mathsf{H}_1(t_0), \mathsf{H}_1(t_1)$ match the first message. Thus, these are also statistical indistinguishable from the simulation.

Accordingly, the transcript generated by the simulator is indistinguishable from the transcript generated by the parties during the real-world execution of the protocol, and thus the adversary cannot distinguish the executions and tell which world it is in.

Finally, when both parties are corrupted, the simulator simply runs the adversary internally which generates the messages for both parties. □

4 Analysis and comparison with the state-of-the-art

In this section, the complexity of the proposed scheme is compared with the state-of-the-art.

The scheme in [BDGM19] is similar to the one presented here, but supports the standard 1-out-of-2 OT definition. Similarly to the proposed scheme, it is supported on the idea of running two key establishment protocols in parallel, such that the receiver only knows the secret to determine one of the keys. In order to achieve UC security, a proof of timely decryption [BDD+17] is required which introduces another message, as well as further calls to the random oracle.

[PVW08] proposed a generic framework for OT, proved UC-secure in the Common Reference String (CRS) model. While it can be instantiated with quantum-safe security assumptions, this framework is based on dual-mode public-key encryption. When considering post-quantum security, this is only known to be achieved under the hardness of the LWE assumption. Therefore, it leads to large parameters that make the scheme impractical. In contrast, it is one of the most efficient UC-secure OTs of the state-of-the-art when instantiated with Elliptic Curve Cryptography (ECC). But, its reliance on ECC makes it insecure in a post-quantum setting. It operates as follows. The receiver uses the CRS to generate a pair of group elements. These elements are combined by the sender with the CRS in two ways to generate two different public-keys that are used for the encryption

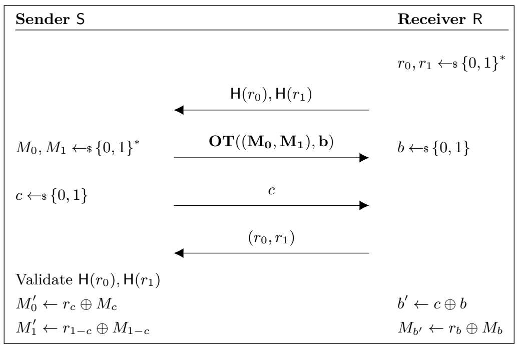

Sender S
Receiver R
$r_0, r_1 \leftarrow\!\$\, \{0,1\}^*$
$\mathsf{H}(r_0), \mathsf{H}(r_1)$
$M_0, M_1 \leftarrow\!\$\, \{0,1\}^*$
$\mathbf{OT}((\mathbf{M_0}, \mathbf{M_1}), \mathbf{b})$
$b \leftarrow\!\$\, \{0,1\}$
$c \leftarrow\!\$\, \{0,1\}$
c
(r_0, r_1)
Validate $\mathsf{H}(r_0), \mathsf{H}(r_1)$
$M_0' \leftarrow r_c \oplus M_c$
$M_1' \leftarrow r_{1-c} \oplus M_{1-c}$
$b' \leftarrow c \oplus b$
$M_{b'} \leftarrow r_b \oplus M_b$

Figure 2: Generic construction of ROT from OT, using a black-box approach.

of M_0 and M_1. Due to the way the scheme is conceived, the receiver only knows the secret-key associated with one of the public-keys, which is used to recover M_b.

Fig. 2 depicts a protocol wherein a ROT is designed supported on a standard OT, following a black-box approach. The security of the resulting protocol depends on the composability of the considered standard OT. In particular, since both [PVW08] and [BDGM19] are proven secure in the UC model for the standard 1-out-of-2 OT definition, this transformation is valid. The ROT is executed as follows. The receiver generates two uniformly random strings, and sends their corresponding commitments to the sender. Then, the base OT is executed for random messages generated by the sender and a random bit b generated by the receiver. Afterwards, the sender generates a random bit c that is sent to the receiver, and the receiver reveals the random strings generated at the beginning. The sender outputs the XOR of the messages it generated and the random strings produced by the receiver. The ordering of the messages is swapped in regards to the base OT if $c = 1$. Similarly, the receiver XORs its received message with the matching string it generated at the beginning, and assigns it the label $b \oplus c$. Since both the messages and their labels are the result of applying XOR operations to values generated at random by the two parties, it is ensured that their distribution cannot be skewed by either of them alone.

A different line of research was followed in [MR19, CO15, CSW20]. Both the protocols in [MR19] and [CSW20] achieve weaker notions of ROT security, tailored for specific OT extensions. The former allows for a malicious sender to choose the two output strings, while a malicious receiver could choose one of the output strings. The latter allows for selective failure attacks by the sender and relaxes the requirements of UC security. While [MR19, CSW20] prove their applicability to specific OT extensions, one cannot, in general, replace ROTs, which have a much wider applicability, with [MR19, CSW20]. As an example, the work of [PRTY19] explicitly uses ROT in its design. Replacing ROT by [MR19, CSW20] would require new security proofs. [CO15] targets the traditional OT setting, but fails to meet the security requirements of UC security. While this allows it to achieve a good performance, since simulators cannot extract a corrupt receiver's choice bit, it is not suitable for many applications, such as OT extensions. Due to their weaker security definitions, the transformation in Fig. 2 is not applicable to [MR19, CO15, CSW20].

A comparison between the computation and communication complexity as well as the security assumptions of the proposed scheme and [PVW08, BDGM19, MR19, CO15, CSW20] can be found in Table 1. Furthermore, the performance for the transformation in Fig. 2 has been considered in the values in parenthesis for [PVW08, BDGM19]. The table includes the number of times the NTT, Gaussian sampling, Random Oracle and symmetric-key encryption functionalities are called. Notice that both the direct and the inverse NTT are associated with the same label (NTT). Similarly, both symmetric-key encryption and decryption are associated with EncDecryption. The complexity of [PVW08, MR19, CO15, CSW20] is evaluated by the number of EC point multiplications required, as well as the number of times the parties need to encode or decode messages as EC group elements.

First, by targeting ROTs instead of OTs, the proposed scheme achieves its functionality without the aforementioned proof of timely decryption. This, in turn, reduces overall complexity when compared with [BDGM19] as it removes entirely the need for symmetric-key encryption. In addition, it removes the need for the last message of the protocol in [BDGM19], and significantly reduces the number of messages when compared with converting [BDGM19, PVW08] to ROTs using a black-box approach. This is particularly important for applications with high-latency, where the communication delay will significantly outweigh the computational delay. These improvements are achieved without the need to change the security assumptions, as in [MR19, CO15, CSW20], which expands its applicability to protocols like [PRTY19].

Second, there has been evidence that RLWE schemes compare favourably to ECC performance-wise [dRVV15]. Indeed, operations over lattices are more likely to benefit from Single Instruction, Multiple Data (SIMD) technologies and multiple instruction issue execution, widely available in modern processors, since they are more regular than usual operations for ECC. These technologies are capable of significantly improving computational efficiency. This improvement in performance comes at the cost of relatively larger messages. The messages of the RLWE schemes included in Table 1 are roughly $\log q$ times larger than those of the ECC schemes, taking as reference the bit-length β of the outputted messages. Nevertheless, the protocol herein proposed reduces the gap in communication complexity relative to [BDGM19]. Future work will focus on closing this gap further.

5 Implementation details

In this section, we describe the techniques used to implement and optimize the performance of the proposed protocol. It should be noted the techniques described here are in general applicable to any protocol supported on the RLWE assumption. In particular, they can be used to accelerate other protocols, like [BDGM19].

The implementation was designed by identifying the operations that were limiting the attained performance, so that the number of times they were instantiated is minimized or that the implementation of the operations themselves be optimized. First, it was identified that Gaussian sampling was a major bottleneck in protocols relying on RLWE. We have used the NFLlib [Qua] library for this sampling. To improve performance, we assume that there is a shared region of memory, or a page if virtual memory is available, that the kernel or some trusted execution environment periodically populates with random data. In this way, the protocol only needs to read data off memory and is unburdened with generating random numbers.

Another bottleneck was related to calls to ROs. ROs were implemented by firstly hashing the inputs, and secondly by using the output of the hash as a seed to a pseudo-random generator. The pseudo-random generator was implemented as a Hash-DRBG [BK12]. This generator was then used to produce the output of the RO. In the case of sampling a polynomial, rejection sampling was used to ensure all coefficients were smaller than the modulus q. This process requires extensive calls to an underlying hash function. We decided to use BLAKE3 [BT] because it is currently the fastest cryptographic hashing algorithm available. Notice that this technique is an adaptation of that employed by widely used protocols such as [KJR16, Appendix B.2]. No fundamental security vulnerability has been found when deploying ROs in this manner.

The final performance bottleneck in the protocol is in the polynomial domain conversion. The NTT is ubiquitous among RLWE implementations. We are using a state-of-the-art implementation in NFLlib [Qua], which supports vector instructions. While the original

Table 1: Theoretical comparison between the proposed scheme and related art. Values in parenthesis refer to the operations/messages added by applying the transformation from Fig. 2 to [BDGM19, PVW08]. Communication cost was estimated based on the following factors. β corresponds to the cyclotomic polynomial degree underpinning RLWE schemes, to $\log p$ for ECs defined in $\mathbb{F}_p$ underpinning ECC schemes, and to the bit-length of the messages outputted by the OT protocol. No point compression is considered (*i.e.* exchanged EC points comprise 2β bits). κ is a security parameter. Small constants (such as the transmission of c in Fig. 2) were ignored.

Scheme	Computation Cost (ROT transform)	Communication Cost (ROT transform)	Security (ROT transform)
This work	**Sender** 4× NTT 3× Gaussian Sampling 5× RO **Receiver** 3× NTT 3× Gaussian Sampling 2× RO	3 messages $\sim 2\beta \log q + 2\beta + 8\kappa$ bits	ROT RLWE ROM UC
[BDGM19]	**Sender** 4× NTT 3× Gaussian Sampling 5× RO 4× EncDecryption **Receiver** 3× NTT 3× Gaussian Sampling 5(+2)× RO 4× EncDecryption	4(+3) messages $\sim 2\beta \log q + 6\beta + 9\kappa$ $(+2\beta + 2\kappa)$ bits	(R)OT RLWE ROM UC
[PVW08] – ECC	**Sender** 8× Point Mult. 2× Message Encoding **Receiver** 3× Point Mult. 1× Message Decoding (2× RO)	2(+3) messages $\sim 12\beta$ $(+2\beta + 2\kappa)$ bits	(R)OT ECC CRS model (ROM) UC
[MR19] – ECC	**Sender** 2× Point Mult. **Receiver** 2× Point Mult. 1× Point Sampling	2 messages $\sim 6\beta$ bits	Endemic OT ECC UC
[CO15] – ECC	**Sender** 3× Point Mult. 2× RO 2× EncDecryption **Receiver** 2× Point Mult. 1× RO 1× EncDecryption	3 messages $\sim 6\beta$ bits	OT ECC Standalone
[CSW20] – ECC	**Sender** 3× Point Mult. 5× RO **Receiver** 2× Point Mult. 4× RO	3 messages $\sim 4\beta + 3\kappa$ bits	Weak ROT ECC Weak UC

library only supports x86 architectures, it has been herein extended to include support for ARM architectures with NEON SIMD extensions. Besides the usage of NFLlib [Qua] to speedup computations in the NTT domain, we also avoid transformations in and out of the domain. *E.g.* we transmit polynomials only in the NTT domain, *e.g.*, p_R^0 and p_S. Further, we consider the outputs of the ROs (H_1 and H_2) to be already in the NTT domain.

6 Experimental results

Prior to integrating the proposed ROT protocol in a PSI framework, the underlying ROT protocol and its sibling OT protocol from [BDGM19] were benchmarked and optimized.

The performance of the proposed implementation of the RLWE ROT and OT protocols is evaluated and compared with related art across several computing architectures, giving insight on their relative scalability. The chosen architectures are four application class ARM machines — Cortex-A7 @ 900MHz, Cortex-A53 @ 1.4GHz, Cortex-A72 @ 1.5GHz, and Apple's M1 @ 3.2GHz — and a server class x86_64 machine — Intel i9-10980XE @ 3GHz. The ARM architectures were selected as they overlap with most embedded consumer electronic devices, *e.g.* on smartwatches and smartphones. The Apple chip was chosen as it is a very close approximation of the architecture found in iPhones and iPads.

The A7 and A53 are in-order (InO) architectures, while the A57 and the M1 are out-of-order (OoO) architectures. These four devices also differ in the wideness of the issue window. The A7 is a partial dual-issue architecture, the A53 is a dual-issue architecture, and the A72 is a triple-issue architecture. While it is known that the M1 device features two types of cores, its issue window architecture was not unveiled (four high power efficient cores and four high performance cores). The results shown herein for this device are running on the high-performance cores. All ARM architectures used in this analysis support NEON vector instructions, and are running in 32-bit mode, with the exception of the M1 device which is running in 64-bit mode. The server class x86_64 platform was selected to verify the maximum performance of the protocol when using a platform with High Performance Computing (HPC) capabilities (*e.g.*, AVX512, and AESNI). The selected x86_64 platform is a 64-bit OoO architecture. The heterogeneity of the devices selected also provides insight into how the vector width impacts the speedup. All ARM devices used support NEON instructions which are 128 bits wide. While, the x86_64 supports vectors 128 bits (SSE4), 256 bits (AVX2), and 512 bits (AVX512) wide. We look to ascertain what is the maximum achievable performance when given a particular point in the power-performance curve, for ROTs and OTs, in a user application scenario by benchmarking multiple implementations in representative devices.

The RLWE ROT and OT implementations are programmed in C++ and use a modified version of NFLlib [Qua][4] in order to support NEON's and AVX512's intrinsics. The proposed OT implementation is compared with state-of-art implementations, namely the OT protocol proposed in [PVW08], which uses OpenSSL as a backend for the elliptic curve arithmetic using the curve NISTP256; [MR19], which also uses NISTP256, supported on libOTe [Rin]; and [CO15], which uses the Twisted Edwards curve described in [BDL$^+$11]. Since [CSW20] has a similar performance to [CO15] (cf. [CSW20, Table 2]), herein, we take the experimental results of [CO15] to be representative of [CSW20] as well. Furthermore, the implementations provided in [Rin] and [CO15] do not support non-x86 architectures. Thus, [MR19, CO15] are not analyzed on ARM devices.

Each program was compiled with GCC 10.1.0 on the x86_64 platform, with GCC 8.3.0 on the A7, A53, and A72 ARM platforms, and with AppleClang 12.0.0 on the M1 platform. In both instances the flags `-O3`, `-march=native`, `-mtune=native` and `-funroll-loops` were used. The testing methodology executes 1k ROTs or OTs 1k times with 100 runs to warm-up the caches in all systems. The parameters used for the RLWE implementation are $N = 512$, $q = 13313$. The hash used for the ROT and OT implementations is BLAKE3 [BT] with vector instructions. The sender and receiver are always executed in a single process and in a single thread, and there is no communication latency. To benchmark, we fix the clock frequency of the architecture and pin the thread to a single core.

[4]The code used in this section can be found on https://github.com/FutureTPM/ROTed.

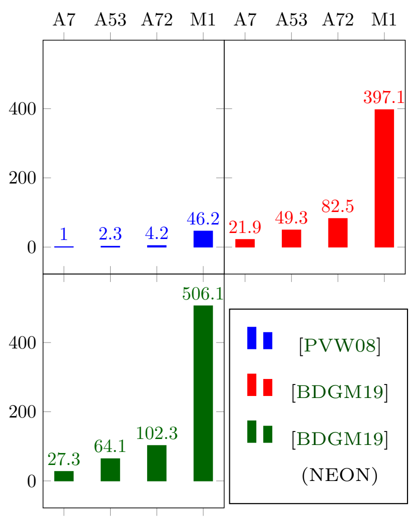


Figure 3: Speedup for the proposed implementation of [BDGM19] and SotA implementation [PVW08] in all ARM devices. The ARM A7 architecture result for [PVW08] is used as the baseline.

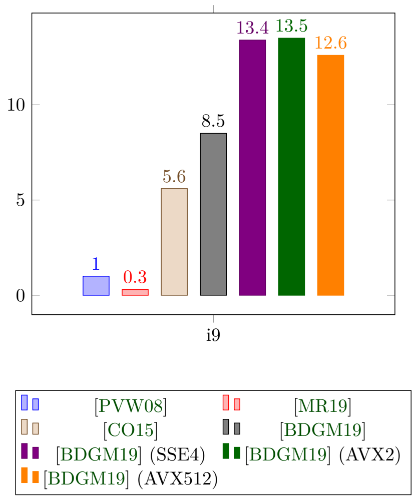


Figure 4: Speedup, for the x86 device, for the proposed implementation of [BDGM19], and SotA [PVW08, MR19, CO15]. The result for [PVW08] is used as the baseline.

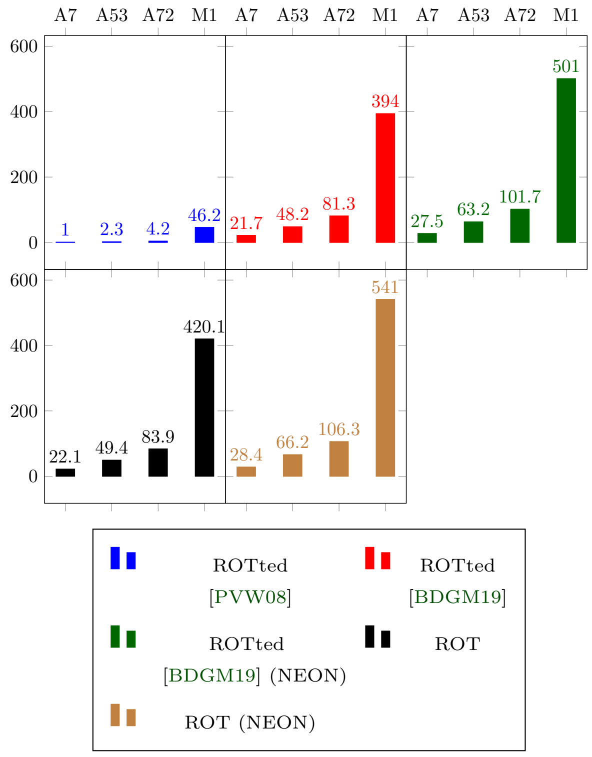
6.1 OT and ROT

Table 2 shows the number of clock cycles (CLK) as well as the time required to execute one OT, and the number of OTs which can be processed in a second for the x86_64 and ARM systems. It should be noted protocols attaining the 1-out-of-2 standard OT as well as Endemic OT in both the UC and standalone models are considered. Figure 3 and Figure 4 show the speedups obtained for each architecture with and without vector instructions. For the ARM systems the implementation from [PVW08] in the ARM A7 architecture is used as the baseline. Similarly, in the x86 system the implementation from [PVW08] is used as the baseline.

The equivalent tables and figures for ROT are Table 3, Fig. 5, and Fig. 6, respectively. Only protocols attaining the same ROT definition as the protocol proposed herein are considered. The transformation described in Fig. 2 was applied to [BDGM19, PVW08] so that they would achieve this security definition.

Figs. 3 and 7 highlight the differences in scalability between the EC-based arithmetic of [PVW08] and the proposed implementation techniques for RLWE-based OTs and ROTs across a wide range of ARM devices. The slowdown in the state-of-the-art implementation from [PVW08] stems from the large number of point multiplications. Profiling this implementation shows that almost 50% of the program is spent performing point multiplications. Indeed, EC arithmetic is inherently sequential. In contrast, RLWE-based cryptosystems are highly amenable to parallelism, and benefit both from the architectural developments that allow for the issue of multiple instructions and from the use of SIMD.

The difference in the ARM architecture backends provides the most speedup to the RLWE implementations. The A72 device, with a OoO backend and wide triple-issue, is 3x and 2x faster than the InO architectures present in the A7 and A53, respectively. The usage of vector instructions provides a modest speedup of 1.30 on ARM (NEON extensions with 128 bits). Apple's M1 outperforms all other ARM devices with a minimum speedup of 3. Even though the M1 and the A72 share the same type of execution backend, there are significant architectural differences between the two. M1's backend is able to issue more



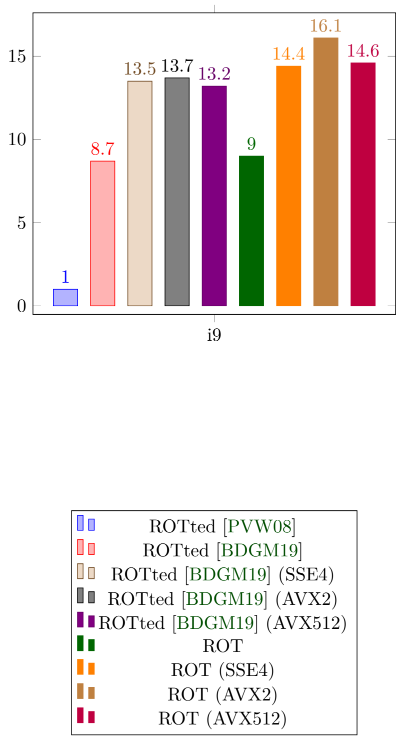


Figure 5: Speedup for the proposed ROT and black-box transformation of state-of-the-art (SotA) OTs [PVW08, BDGM19] into ROTs in all ARM devices. The ARM A7 architecture result for the ROTted [PVW08] is used as the baseline.

Figure 6: Speedup for the x86 device, for the proposed ROT, and black-box transformation of SotA OTs [PVW08, BDGM19] into ROTs. ROTted [PVW08] is the baseline.

instructions in a single clock cycle and to have more in-flight instructions than the A72. Similarly to the other ARM devices, the usage of NEON's vector instructions provides a smaller speedup when compared to improving the execution backend. The speedups obtained from using vector instruction are around 30% whereas the change of the execution backend provides speedups larger than 100%. This suggests that the usage of an OoO execution backend provides a greater speedup than the addition of wider vectors.

Figs. 4 and 6 provide a more detailed comparison between the performance of the proposed implementation techniques for RLWE-based OTs and ROTs and the state-of-the-art in a x86 device. They also allow for a deeper analysis of the impact of vectorization. The main difference between the vector extensions to the x86 Instruction Set Architecture (ISA) is their bit width. The AVX2 ISA supports 256-bit vectors, while the SSE4 ISA supports 128-bit vectors. The difference in speedup between the AVX2 and the SSE4 implementations is 12%, suggesting, similarly to the ARM platforms, that the usage of an OoO execution backend provides a greater speedup than the addition of wider vectors. This conclusion also goes inline with the described bottlenecks in the previous subsection. Among the bottlenecks referred in Section 5, NTT is the one that least impacts performance. The RLWE AVX512 implementation, which employs the widest available vector unit with 512 bits, shows a slowdown when compared with the AVX2 implementation. This is because in some cases we are not able to fill the vector, thus we need to use a smaller vector size or use the serial implementation. Therefore, length checks had to be added in the NTT loop in order to call the correct functions, leading to an increased number of

Table 2: Performance evaluation of [BDGM19] using the proposed implementation techniques, and state-of-the-art (SotA) implementations [PVW08, MR19, CO15].

ARM Cortex-A7 @ 900MHz	**Security**	**CLK (k)**	**Time (μs)**	**OTs/s**
SotA [PVW08]	OT/UC	18226.8	20252	50
RLWE OT (Serial)	OT/UC	835.47	928.3	1078
RLWE OT (NEON)	OT/UC	669.33	743.7	1345
ARM Cortex-A53 @ 1.4GHz				
SotA [PVW08]	OT/UC	12826.8	9162	110
RLWE OT (Serial)	OT/UC	575.82	411.3	2432
RLWE OT (NEON)	OT/UC	442.68	316.2	3163
ARM Cortex-A72 @ 1.5GHz				
SotA [PVW08]	OT/UC	7368	4912	204
RLWE OT (Serial)	OT/UC	368.4	245.6	4072
RLWE OT (NEON)	OT/UC	97.15	198.1	5048
Apple M1 @ 3.2GHz				
SotA [PVW08]	OT/UC	1405.1	439.1	2278
RLWE OT (Serial)	OT/UC	163.2	51	19608
RLWE OT (NEON)	OT/UC	128.3	40.1	24938
Intel i9-10980XE @ 3GHz				
SotA [PVW08]	OT/UC	1278.6	426.2	2347
SotA [MR19]	End.OT/UC	5214	1738	576
SotA [CO15]	OT/Stand.	229.8	76.6	13055
RLWE OT (Serial)	OT/UC	150.6	50.2	19921
RLWE OT (SSE4)	OT/UC	95.7	31.9	31348
RLWE OT (AVX2)	OT/UC	95.4	31.8	31447
RLWE OT (AVX512)	OT/UC	101.7	33.9	29499

missed branch predictions. Even though the NTT is not a major bottleneck, it remains a hot loop. The additional branches in the hot loop have a significant misprediction rate which cause the slowdown.

The previous discussion shows that the proposed RLWE implementation is more portable across devices than related art. In fact, the proposed RLWE-based implementations outperform state-of-the-art implementations across all devices in all the aforementioned figures.

The RLWE implementation for OT uses 180.4KiB, a similar amount of memory to the [PVW08] implementation, which uses 156.5KiB. Therefore, the RLWE implementation uses 16% (23.9KiB) more memory than the state-of-the-art implementation. The size of the polynomials and the auxiliary data required to perform its arithmetic are major contributors to the memory increase. This is aligned with previous results showing that, while RLWE achieves lower latency and seems resistant to quantum computing, it requires more memory consumption [FMS20]. Nevertheless, this difference in memory is negligible in most of today's devices total memory. In contrast, the Endemic OT of [MR19] uses 13.1KiB. The memory difference for the RLWE implementation is steeper, 1277% (167.3 KiB), when compared with the state-of-the-art implementation in [MR19]. Converting an Endemic OT to a ROT would require adding further communications rounds, using a transform similar in spirit but more complicated than the one in Fig. 2. This suggests it might be possible to reduce memory consumption at the cost of introducing further communication rounds, which might be necessary for IoT devices with restrictive memory requirements. Finally, this memory consumption analysis is similar to both the ROT and OT protocols.

A speedup comparison between the proposed ROT and the OT scheme from [BDGM19],

Table 3: ROT from Fig. 1 implementation results for all systems.

ARM Cortex-A7 @ 900MHz	**CLK (k)**	**Time (μs)**	**ROTs/s**
SotA [PVW08] ROTted	18258	20287	50
[BDGM19] ROTted (Serial)	843	936.6	1068
[BDGM19] ROTted (NEON)	666	739.7	1352
RLWE ROT (Serial)	829.08	921.2	1086
RLWE ROT (NEON)	644.94	716.6	1396
ARM Cortex-A53 @ 1.4GHz			
SotA [PVW08] ROTted	12864.6	9189	109
[BDGM19] ROTted (Serial)	589.3	420.9	2376
[BDGM19] ROTted (NEON)	450	321.4	3112
RLWE ROT (Serial)	574.98	410.7	2435
RLWE ROT (NEON)	429.52	306.8	3260
ARM Cortex-A72 @ 1.5GHz			
SotA [PVW08] ROTted	7378.5	4919	204
[BDGM19] ROTted (Serial)	374.6	249.7	4005
[BDGM19] ROTted (NEON)	299.4	199.6	5011
RLWE ROT (Serial)	362.7	241.8	4136
RLWE ROT (NEON)	286.5	191	5236
Apple M1 @ 3.2GHz			
SotA [PVW08] ROTted	1407.7	439.9	2274
[BDGM19] ROTted (Serial)	164.8	51.5	19418
[BDGM19] ROTted (NEON)	129.6	40.5	24692
RLWE ROT (Serial)	154.6	48.3	20704
RLWE ROT (NEON)	120	37.5	26667
Intel i9-10980XE @ 3GHz			
SotA [PVW08] ROTted	1310.7	436.9	2289
[BDGM19] ROTted (Serial)	151.5	50.5	19802
[BDGM19] ROTted (SSE4)	97.2	32.4	30865
[BDGM19] ROTted (AVX2)	96.3	32.1	31153
[BDGM19] ROTted (AVX512)	99.6	33.2	30121
RLWE ROT (Serial)	147	49	20409
RLWE ROT (SSE4)	91.2	30.4	32895
RLWE ROT (AVX2)	81.6	27.2	36765
RLWE ROT (AVX512)	90.3	30.1	33223

for all ARM devices and using the same optimization techniques described in section 5, is included in Figure 7. The ROT protocol, as expected from the theoretical construction and observed in Table 2 and Table 3, is faster than the OT protocol. In this case, since we are not measuring the latency between parties, the difference in execution times is due to the OT requiring symmetric encryption, which the ROT does not. If latency were accounted for, the difference would be even greater, as the OT requires the transmission of one more message than the ROT.

6.2 PSI with the proposed ROTs

The proposed ROT was integrated in the PaXos PSI framework [PRTY20a] to measure the ROT impact in an application scenario. Since the framework is for x86_64, results are only provided for that platform. The parameters used in these tests are `fieldSize` = 231 and `hashSize` = 2048, corresponding to the bit-length of the code-words of the underlying OT extension and the cardinality of the sets being intersected, respectively. We follow the same testing methodology as in the previous subsection, the receiver and the sender

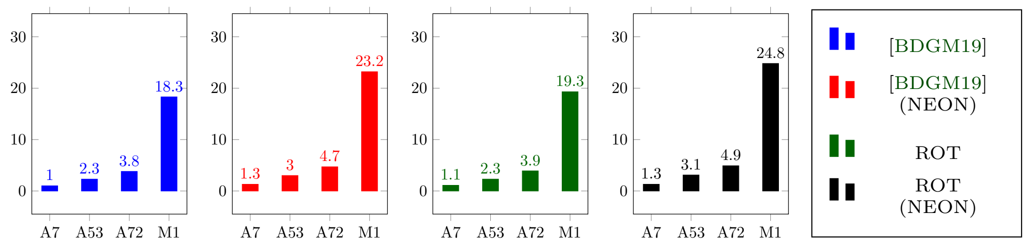


Figure 7: Speedup for the proposed ROT and [BDGM19] in all ARM devices. The ARM A7 architecture result for [BDGM19] is used as the baseline.

Table 4: PSI time for the x86 server architecture.

Intel i9-10980XE @ 3GHz	**Time (ms)**
SotA [PVW08] ROTted	932
[BDGM19] ROTted (Serial)	328
[BDGM19] ROTted (AVX2)	304
[BDGM19] ROTted (AVX512)	318
RLWE ROT (Serial)	166
RLWE ROT (AVX2)	142
RLWE ROT (AVX512)	151

Table 5: PSI peak memory for the x86_64 system.

	Peak Memory (MiB)
Sender	15.5
Receiver	11.8

are running in the same machine. The only exception is that the PaXos PSI framework communicates solely through sockets, thus the latency of setting up a TCP connection and its protocol latency are also measured. The results in Table 4 show the time taken to perform one PSI, Figure 8 shows the speedup between the different ROT implementations, and Table 5 shows the peak memory usage for the sender and the receiver.

Similarly to the conclusions from the ROT result analysis, the AVX2 implementation is the fastest and the AVX512 implementation shows a slowdown. The vector implementations show the same speedup when compared to the serial implementation. The proposed ROT use in a PSI provides a 6.6x speedup when compared with the ROTted version of [PVW08] and a 2.1x speedup with the ROTted version of the sibling OT [BDGM19]. The speedup gains are significative, however, it should be noted that the ROT transformation of the original protocols contributed to the slowdowns. The ROT transformation adds 3 messages to the protocol, 4 hashes, 3 random samples, and 4 XORs. In the current testing setup,

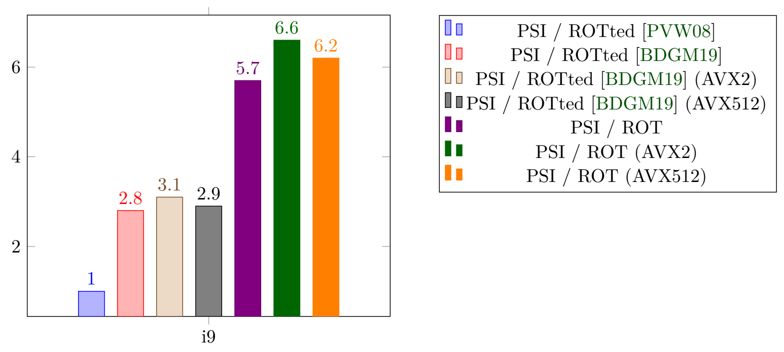


Figure 8: PSI speedup results for the x86_64 system.

message latency is not considered. Both the sender and receiver run in the same core and in the same machine. Therefore, the cost of the extra messages is reflected in these tests as additional system calls in order to provide Inter Process Communication (IPC) between the sender and the receiver. As such, the speedup gains in the PSI using the proposed ROT result from the compounded effect of all the optimizations performed and the reduced number of messages.

The memory requirements by the PaXos PSI framework far exceed those of the proposed ROT. This demand is rooted in the pseudo-random number generator (PRNG) and the linear code used, neither having any relation to the ROT used. Therefore, the addition of the proposed ROT to PSI protocols can be done with no associated cost of improving the computing platform, while providing better performance and achieving greater security.

7 Conclusions

This paper proposes a UC-secure ROT protocol from the RLWE assumption in the ROM. Not only does RLWE allow for the exploitation of efficient arithmetic supported on the ring structure, but it was also shown that by considering ROT instead of the traditional OT, further performance improvements are achieved. Moreover, ROT can be used to support OT extensions with wide applicability.

Performance-wise, it is shown, through extensive experimental evaluation, that the proposed ROT compares favourably to the state-of-the-art OTs, based on RLWE and ECC. The arithmetic associated with lattices is more prone to parallelization than ECC, and the usage of vector instructions provides on average a 40% speedup for the proposed protocol. Further, from the experimental results, the proposed protocol is amenable to a high level of instruction level parallelism, as the usage of an OoO backend provides a minimum of 2x speedup resulting in up to 37k ROTs/s for the Intel server-class processor and up to 5k ROTs/s in an ARM application-class processor. Therefore, our proposal is at least one order of magnitude faster than the state-of-the-art, and is suitable for a wide range of architectures in embedded systems, IoT, desktops and servers. Finally, it is shown that the proposed protocol is of practical interest by integrating it in a PSI framework with applications in contact discovery, remote diagnosis, contact tracing, among others. The usage of the proposed ROT in a PSI application is up to 6 times faster than related art.

It is clear from the extensive performance analysis provided that there is still more room for performance improvements. Moreover, the devices used in the experimental analysis omit ultra-low power devices, many of which do not possess OoO execution backends, and would benefit from the usage of Domain Specific Accelerators (DSAs). Therefore, future work will focus on designing and implementing the proposed DSAs and performing a thorough performance analysis of such devices, *e.g.*, ARM Cortex-M and RISC-V.

Acknowledgments

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and UIDB/50008/2020 (Instituto de Telecomunicações via actions QuRUNNER, QUESTS) and Projects QuantumMining POCI-01-0145-FEDER-031826, PREDICT PTDC/CCI-CIF/29877/2017 and QuantumPrime PTDC/EEI-TEL/8017/2020. The authors thank the reviewers for their support in improving this paper. The authors also thank Rafael Gonçalves for providing access to a Raspberry Pi 3 for benchmarking. PB thanks the support from DP-PMI and FCT (Portugal) through the grant PD/BD/135181/2017, LF the support from FCT (Portugal) through the grant SFRH/BD/145477/2019, and MG the support from DP-PMI and FCT (Portugal) through the grant PD/BD/135182/2017.

References

[ABC$^+$20] Nick Angelou, Ayoub Benaissa, Bogdan Cebere, William Clark, Adam James Hall, Michael A. Hoeh, Daniel Liu, Pavlos Papadopoulos, Robin Roehm, Robert Sandmann, Phillipp Schoppmann, and Tom Titcombe. Asymmetric private set intersection with applications to contact tracing and private vertical federated machine learning, 2020. arXiv:2011.09350.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-03356-8_35.

[BDD$^+$17] Paulo S. L. M. Barreto, Bernardo David, Rafael Dowsley, Kirill Morozov, and Anderson C. A. Nascimento. A framework for efficient adaptively secure composable oblivious transfer in the ROM. Cryptology ePrint Archive, Report 2017/993, 2017. https://eprint.iacr.org/2017/993.

[BDF$^+$11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-25385-0_3.

[BDGM19] Pedro Branco, Jintai Ding, Manuel Goulão, and Paulo Mateus. A framework for universally composable oblivious transfer from one-round key-exchange. In Martin Albrecht, editor, *Cryptography and Coding*, pages 78–101, Cham, 2019. Springer International Publishing.

[BDL$^+$11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142. Springer, 2011. URL: https://www.iacr.org/archive/ches2011/69170125/69170125.pdf, doi:10.1007/978-3-642-23951-9_9.

[BK12] Elaine B. Barker and John M. Kelsey. Sp 800-90a. recommendation for random number generation using deterministic random bit generators. Technical report, Gaithersburg, MD, USA, 2012.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 498–507, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1315245.1315307.

[BT] BLAKE3-Team. BLAKE3. URL: https://github.com/BLAKE3-team/BLAKE3.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press. doi:10.1109/SFCS.2001.959888.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany. doi:10.1007/3-540-44647-8_2.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230 of *Lecture Notes in Computer Science*, pages 40–58, Guadalajara, Mexico, August 23–26, 2015. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-22174-8_3.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Blazing fast ot for three-round uc ot extension. Cryptology ePrint Archive, Report 2020/110, 2020. https://eprint.iacr.org/2020/110.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. Cryptology ePrint Archive, Report 2018/579, 2018. https://eprint.iacr.org/2018/579.

[dRVV15] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of ring-LWE encryption. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 339–344, 2015. doi:10.7873/DATE.2015.0378.

[DXL12] Jintai Ding, Xiang Xie, and Xiaodong Lin. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688, 2012. http://eprint.iacr.org/2012/688.

[FMS20] Luís Fiolhais, Paulo Martins, and Leonel Sousa. Software emulation of quantum resistant trusted platform modules. In *2020 International Conference on Security and Cryptography (SECRYPT)*, pages 477–484, 01 2020.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press. doi:10.1145/28395.28420.

[HL17] Eduard Hauck and Julian Loss. Efficient and universally composable protocols for oblivious transfer from the CDH assumption. Cryptology ePrint Archive, Report 2017/1011, 2017. http://eprint.iacr.org/2017/1011.

[HOSS18] Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. TinyKeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 3–33, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-96878-0_1.

[IKN$^+$19] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. Cryptology ePrint Archive, Report 2019/723, 2019. https://eprint.iacr.org/2019/723.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-45146-4_9.

[ISMG20] Cornelius Ihle, Moritz Schubotz, Norman Meuschke, and Bela Gipp. A first step towards content protecting plagiarism detection. *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*, Aug 2020. URL: http://dx.doi.org/10.1145/3383583.3398620, doi:10.1145/3383583.3398620.

[KJR16] B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2 . RFC 8017, November 2016. URL: https://tools.ietf.org/html/rfc8017.

[KMO90] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. Minimum resource zero-knowledge proofs. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 545–546, New York, NY, 1990. Springer New York.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-13190-5_1.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 35–54, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-38348-9_3.

[MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 309–326. ACM Press, November 11–15, 2019. doi:10.1145/3319535.3354210.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 381–396, San Francisco, CA, USA, February 14–17, 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-52153-4_22.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-26954-8_13.

[PRTY20a] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 739–767, Cham, 2020. Springer International Publishing.

[PRTY20b] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. Cryptology ePrint Archive, Report 2020/193, 2020. https://eprint.iacr.org/2020/193.

[PSSW09] B. Pinkas, T. Schneider, N.P. Smart, and S. Williams. Secure two-party computation is practical. Cryptology ePrint Archive, Report 2009/314, 2009. https://eprint.iacr.org/2009/314.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-85174-5_31.

[Qua] QuarksLab. NTT-based fast lattice library. URL: https://github.com/quarkslab/NFLlib.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press. doi:10.1145/1060590.1060603.

[Rin] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[Yao82] Andrew Yao. Protocols for secure computations (extended abstract). In *FOCS 1982*, 1982.