More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries[⋆]

Gilad Asharov[1], Yehuda Lindell[2], Thomas Schneider[3], and Michael Zohner[3]

[1] The Hebrew University of Jerusalem, Israel
asharov@cs.huji.ac.il
[2] Bar-Ilan University, Israel
lindell@biu.ac.il
[3] TU Darmstadt, Darmstadt, Germany
{thomas.schneider,michael.zohner}@ec-spride.de

Abstract. Oblivious transfer (OT) is one of the most fundamental primitives in cryptography and is widely used in protocols for secure two-party and multi-party computation. As secure computation becomes more practical, the need for practical large scale oblivious transfer protocols is becoming more evident. Oblivious transfer extensions are protocols that enable a relatively small number of "base-OTs" to be utilized to compute a very large number of OTs at low cost. In the semi-honest setting, Ishai et al. (CRYPTO 2003) presented an OT extension protocol for which the cost of each OT (beyond the base-OTs) is just a few hash function operations. In the malicious setting, Nielsen et al. (CRYPTO 2012) presented an efficient OT extension protocol for the setting of active adversaries, that is secure in the random oracle model.

In this work, we present an OT extension protocol for the setting of malicious adversaries that is more efficient and uses less communication than previous works. In addition, our protocol can be proven secure in both the random oracle model, and in the standard model with a type of correlation robustness. Given the importance of OT in many secure computation protocols, increasing the efficiency of OT extensions is another important step forward to making secure computation practical.

Keywords: Oblivious transfer extensions, concrete efficiency, secure computation

[⋆] This work was partially supported by the European Union's Seventh Framework Program (FP7/2007-2013) grant agreement n. 609611 (PRACTICE). The first author is supported by the Israeli Centers of Research Excellence (I-CORE) Program (Center No. 4/11). The second is supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC consolidators grant agreement n. 615172 (HIPS). The third and fourth authors are supported by the DFG as part of project E3 within the CRC 1119 CROSSING, by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, and by the Hessian LOEWE excellence initiative within CASED.

1 Introduction

1.1 Background

Oblivious Transfer (OT), introduced by Rabin [33], is a fundamental cryptographic protocol involving two parties, a sender and a receiver. In the most commonly used 1-out-of-2 version [9], the sender has a pair of messages (x_0, x_1) and the receiver has a selection bit r; at the end of the protocol the receiver learns x_r (but nothing about x_{1-r}) and the sender learns nothing at all about r. Oblivious transfer is a fundamental tool for achieving secure computation, and plays a pivotal role in the Yao protocol [35] where OT is needed for every bit of input of the client, and in the GMW protocol [12] where OT is needed for every AND gate in the Boolean circuit computing the function.

Protocols for secure computation provide security in the presence of adversarial behavior. A number of adversary models have been considered in the literature. The most common adversaries are: *passive* or *semi-honest adversaries* who follow the protocol specification but attempt to learn more than allowed by inspecting the protocol transcript, and *active* or *malicious adversaries* who run any arbitrary strategy in an attempt to break the protocol. In both these cases, the security of a protocol guarantees that nothing is learned by an adversary beyond its legitimate output. Another notion is that of security in the presence of *covert adversaries*; in this case the adversary may follow any arbitrary strategy, but is guaranteed to be caught with good probability if it attempts to cheat. The ultimate goal in designing efficient protocols is to construct protocols that are secure against strong (active or covert) adversaries while adding very little overhead compared to the passive variant. In our paper, we focus primarily on the case of active adversaries, but also provide a variant for covert security.

OT extensions. As we have mentioned, OT is used extensively in protocols for secure computation. In many cases, this means several millions of oblivious transfers must be run, which can become prohibitively expensive. Specifically, the state-of-the-art protocol for achieving OT with security in the presence of active adversaries of [31] achieves approximately 350 random OTs per second on standard PCs. However, a million OTs at this rate would take over 45 minutes. In order to solve this problem, OT extensions [4] can be used. An OT extension protocol works by running a small number of *"base-OTs"* depending on the security parameter (e.g., a few hundred) that are used as a base for obtaining many OTs via the use of cheap symmetric cryptographic operations only. This is conceptually similar to public-key encryption where instead of encrypting a large message using RSA, which would be too expensive, a hybrid encryption scheme is used such that the RSA computation is only carried out to encrypt a symmetric key, which is then used to encrypt the large message. Such an OT extension can be achieved with extraordinary efficiency; specifically, the protocol of [16] for passive adversaries requires only three hash function computations per OT (beyond the initial base-OTs). In [1], by applying additional algorithmic and cryptographic optimizations, the cost of OT extension for passive adversaries is so low that essentially the communication is the bottleneck. To be concrete, 10,000,000 OTs on random inputs (which suffices for many applications) can be carried out in just 2.62 seconds over a LAN with four threads [1].

For active adversaries, OT extensions are somewhat more expensive. Prior to this work, the best protocol known for OT extensions with security against active adversaries was introduced by [30]. The computational cost of the protocol is due to the number of base-OTs needed for obtaining security, the number of symmetric operations (e.g., hash function computations) needed for every OT in the extension, and the bandwidth. Relative to the passive OT extension of [16], the run-time of [30] is approximately 4 times longer spent on the base-OTs, 1.7 times the cost for each OT in the extension, and 2.7 times the communication. Asymptotically, regarding the number of base-OTs, for security parameter κ (e.g., $\kappa = 128$), it suffices to run κ base-OTs in the passive case. In contrast, [30] require $\lceil \frac{8}{3}\kappa \rceil$ base-OTs.

Applications of OT for malicious adversaries. Most prominently, OT is heavily used in today's most efficient protocols for *secure computation* that allow two or more parties to securely evaluate a function expressed as Boolean circuit on their private inputs. Examples include Yao's garbled circuits-based approaches such as [22, 24, 32, 19, 11, 34, 25, 14, 10] where OTs are needed for each input, or the Tiny-OT [30, 21] and MiniMac protocols [6, 5] where OTs are needed for each AND gate. Additional applications include the *private set intersection* protocol of [7] which is based purely on OT, and the Yao-based *zero-knowledge* protocol of [17] which allows a party to prove in zero-knowledge a predicate expressed as Boolean circuit, and needs one OT per bit of the witness.

In many of the above applications, the number of oblivious transfers needed can be huge. For instance, for many applications of practical interest, the two-party and multiparty protocols of [30, 21, 6, 5, 7] can require several hundred millions of OTs, making the cost of OT the bottleneck in the protocol. Concretely, the current implementations of secure computation in the malicious setting requires $\sim 2^{19}$ OTs for the AES circuit and $\sim 2^{30}$ OTs for the PSI circuit (Sort-Compare-Shuffle), see full version [2] for further details. Thus, improved OT extensions immediately yield faster two-party and multi-party protocols for secure computation.

1.2 Our Contributions

In this paper, we present a new protocol for OT extensions with security in the presence of malicious adversaries, which outperforms the most efficient existing protocol of [30]. We follow the insights of prior work [1, 11], which show that the bottleneck for efficient OT extension is the communication, and focus on decreasing the communication at the cost of slightly increased computation. Furthermore, our protocol can be instantiated with different parameters, allowing us to tradeoff communication for computation. This is of importance since when running over a LAN the computation time is more significant than when running over a WAN where the communication cost dominates. We implement and compare our protocol to the semi-honest protocol of [16] (with optimizations of [18, 1]) and the malicious protocol of [30] (with optimizations of [11]). As can be seen from the summary of our results given in Table 1, our actively secure protocol performs better than the previously fastest protocol of [30] running at under 60% the cost of the base-OTs of [30], 70% of the cost of each OT in the extension, and 55% of the communication in the local setting. Due to the lower communication, the improvement of our protocol over [30] in the cloud setting (between US East and Europe and thus with higher latency), is even greater with approximately 45% of the time of the base-OTs and 55% of the time for each OT in the extension.

Comparing our protocol to the passive OT extension of [16], our actively secure protocol in the local (LAN) setting costs only 133% more run-time in the base-OTs, 20% more run-time for each OT in the extension, and 50% more communication. In the cloud setting, the cost for each OT in the extension is 63% more than [16] (versus 293% more for [30]). Finally, we obtain covert security at only a slightly higher cost than passive security (just 10% more for each OT in the extension in the local setting, and 30% more in the cloud setting). Our protocol reduces the number of base-OTs that are required to obtain malicious security from $\frac{8}{3}\kappa$ for [30] to $\kappa + \epsilon\rho$, where ρ is the statistical security parameter (e.g., ρ=40) and $\epsilon \geq 1$ is a parameter for trading between computation and communication. To be concrete, for κ=128-bit security, our protocol reduces the number of base-OTs from 342 to 190 in the local and to 174 in the cloud setting.

Prot.	**Security**	**Run-Time**		**Communication**	
		Local	**Cloud**	**Local**	**Cloud**
[16]	passive	0.3s+1.07$\mu s \cdot t$	0.7s+4.24$\mu s \cdot t$	4KB+128bit $\cdot\, t$	
This	covert	0.6s+1.18$\mu s \cdot t$	1.2s+5.48$\mu s \cdot t$	21KB+166bit $\cdot\, t$	
[20]	active	-	-	42KB+106,018bit $\cdot\, t$	
[31]	active	2975.32$\mu s \cdot t$	4597.27$\mu s \cdot t$	0.3KB+1,024bit $\cdot\, t$	
[30]	active	1.2s+1.82$\mu s \cdot t$	2.9s+12.43$\mu s \cdot t$	43KB+342bit $\cdot\, t$	
This	active	0.7s+1.29$\mu s \cdot t$	1.3s+6.92$\mu s \cdot t$	24KB+191bit $\cdot\, t$	22KB+175bit $\cdot\, t$

Table 1. Run-time and communication for t random OT extensions with κ=128-bit security (amortized over 2^{26} executions; [31] amortized over 2^{14} executions). 1KB= 8,192bit.

In addition to being more efficient, we can prove the security of a variant of our protocol with a version of correlation robustness (where the secret value is chosen with high min-entropy, but not necessarily uniformly), and do not require a random oracle (see §3.3). In contrast, [30] is proven secure in the random oracle model. [4] Our implementation is available online at `http://encrypto.de/code/OTExtension` and was integrated into the SCAPI library [8] available at `https://github.com/cryptobiu/scapi`.

[4] It is conjectured that the [30] OT can be proven secure without a random oracle, but this has never been proven.

1.3 Related Work

The first efficient OT extension protocol for semi-honest adversaries was given in [16]. Improvements and optimizations to the protocol of [16] were given in [18, 1].

Due to its importance, a number of previous works have tackled the question of OT extensions with security for malicious/active adversaries. There exist several approaches for achieving security against active adversaries for OT extensions. All of the known constructions build on the semi-honest protocol of [16], and add *consistency checks* of different types to the OT extension protocol, to ensure that the receiver sent consistent values. (Note that in [16], the sender cannot cheat and so it is only necessary to enforce honest behavior for the receiver.)

The first actively-secure version of OT extension used a cut-and-choose technique and was already given in [16]. This cut-and-choose technique achieves a security of 2^{-n} by performing n parallel evaluations of the basic OT extension protocol.

This was improved on by [29, 13], who show that active security can be achieved at a much lower cost. Their approach works in the random oracle model and ensures security against a malicious receiver by adding a low-cost check per extended OT, which uses the uncertainty of the receiver in the choice bit of the sender. As a result, a malicious receiver who wants to learn p choice bits of the sender risks being caught with probability 2^{-p}. However, this measure allows a malicious sender to learn information about the receiver's choice bits. They prevent this attack by combining $S \in \{2, 3, 4\}$ OTs and ensuring the security of one OT by sacrificing the remaining $S - 1$ OTs. Hence, their approach adds an overhead of at least $S \geq 2$ compared to the semi-honest OT extension protocol of [16] for a reasonable number of OTs (with $S = 2$ and approximately 10^7 OTs, they achieve security except with probability 2^{-25}, cf. [29]). However, the exact complexity for this approach has not been analyzed.

An alternative approach for achieving actively-secure OT extension was given in [30]. Their approach also works in the random oracle model but, instead of performing checks per extended OT as in [29, 13], they perform consistency checks per base-OT. Their consistency check method involves hashing the strings that are transferred in the base-OTs and is highly efficient. In their approach, they ensure the security of a base-OT by sacrificing another base-OT, which adds an overhead of factor 2. In addition, a malicious receiver is able to learn p choice bits of the sender with probability 2^{-p}. [30] shows that this leakage can be tolerated by increasing the number of base-OTs from κ to $\lceil \frac{4}{3}\kappa \rceil$. Overall, their approach increases the number of base-OTs that has to be performed by a *multiplicative factor* of $\frac{8}{3}$. The [30] protocol has been optimized and implemented on a GPU in [11]. We give a full description of the [30] protocol with optimizations of [11] in Appendix §B.

An approach for achieving actively-secure OT extension that works in the standard model has recently been introduced in [20]. Their approach achieves less overhead in the base-OTs at the expense of substantially more communication during the check routine (cf. Table 1), and is therefore considerably less efficient. Nevertheless, we point out that the work of [20] is of independent interest since it is based on the original correlation robustness assumption only.

Since it is the previous best, we compare our protocol to that of [30]. Our approach reduces the number of base-OTs by removing the “sacrifice” step of [30] (where one out of every 2 base-OTs are opened) but increases the workload in the consistency check routine. Indeed, we obtain an additive factor of a statistical security parameter, instead of the multiplicative increase of [30]. This can be seen as a trade-off between reducing communication through fewer base-OTs while increasing computation through more work in the consistency check routine. We empirically show that this results in a more efficient actively secure OT extension protocol, which only has 20% more time and 50% more communication than the passively secure OT extension protocol of [16] in the local setting.

The above works all consider the concrete efficiency of OT extensions. The theoretical feasibility of OT extensions was established in [4], and further theoretical foundations were laid in [26].

2 Preliminaries

2.1 Notation

Our protocol uses a computational (symmetric) security parameter κ and a statistical security parameter ρ. Asymptotically, this means that our protocols are secure for any adversary running in time $\text{poly}(\kappa)$, except with probability $\mu(\kappa) + 2^{-\rho}$. (Formally, the output distribution of a real protocol execution can be distinguished from the output distribution of an ideal execution of the OT functionality with probability at most $\mu(\kappa) + 2^{-\rho}$. See [23] for a formal definition of secure computation with both a statistical and computational security parameter.) In our experiments we set $\kappa = 128$ and $\rho = 40$, which is considered to be secure beyond 2020[5].

2.2 Oblivious Transfer

Oblivious transfer (OT) was first introduced by Rabin [33] as a function where a receiver receives a message, sent by a sender, with probability 1/2, while the sender remains oblivious whether the message was received. It was later redefined to the functionality more commonly used today by [9], where a sender inputs two messages (x_0, x_1) and the receiver inputs a choice bit r and obliviously receives x_r without learning any information about x_{1-r}. Formally, the 1-out-of-2 OT functionality on n bit strings is defined as $OT_n((x_0, x_1), r) = (\lambda, x_r)$ where λ denotes the empty string and $x_0, x_1 \in \{0,1\}^n$. In this paper we focus on the general (and most applicable) functionality, which is equivalent to m invocations of the 1-out-of-2 OT functionality on n bit strings. That is, the sender holds as

[5] According to the summary of cryptographic key length recommendations at http://keylength.com.

input m pairs of n-bit strings (x_j^0, x_j^1) for $1 \leq j \leq m$ and the receiver holds m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$. The output of the receiver is $(x_1^{r_1}, \ldots, x_m^{r_m})$ while the sender has no output. We denote this functionality as $m \times OT_n$. The parties are called sender P_S and receiver P_R.

Several protocols for OT based on different cryptographic assumptions and attacker models were introduced. Most notable are the passive-secure OT protocol of [28] and the active-secure OT protocol of [31], which are among the most efficient today. However, the impossibility result of [15] showed that OT protocols require costly asymmetric cryptography, which greatly limits their efficiency.

2.3 OT Extension

In his seminal work, Beaver [4] introduced *OT extension* protocols, which extend few costly *base-OTs* using symmetric cryptography only. While the first construction of [4] was inefficient and mostly of theoretical interest, the protocol of [16] showed that OT can be extended efficiently and with very little overhead.

Recently, the passively secure OT extension protocol of [16] was improved by [18, 1] who showed how the communication from P_R to P_S can be reduced by a factor of two. Furthermore, [1] implemented and optimized the protocol and demonstrated that the main bottleneck for semi-honest OT extension has shifted from computation to communication. We give the passively secure OT extension protocol of [16] with optimizations from [1, 18] in Protocol 1.

2.4 On the Malicious Security of [16]

The key insight to understanding how to secure OT extension against malicious adversaries is to understand that a malicious party only has very limited possibilities for an attack. In fact, the original OT extension protocol of [16] already provides security against a malicious P_S. In addition, the only attack for a malicious P_R is in Step 2a of Protocol 1, where P_R computes and sends $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ (cf. [16]). A malicious P_R could choose a different $\mathbf{r}$ for each $\mathbf{u}^i$ (for $1 \leq i \leq \ell$), and thereby extract P_S's choice bits $\mathbf{s}$. Hence, malicious security can be obtained if P_R can be forced to use the same choice bits $\mathbf{r}$ in all messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$.

3 Our Protocol

All we add to the semi-honest protocol (Protocol 1) is a consistency check for the values $\mathbf{r}$ that are sent in Step 2a, and increase the number of base-OTs. Let $\mathbf{r}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{u}^i$, i.e., the value that is implicitly defined by $\mathbf{u}^i$. We observe that if the receiver P_R uses the same choice bits $\mathbf{r}^i$ and $\mathbf{r}^j$ for some distinct $i, j \in [\ell]^2$, they cancel out when computing their XOR, i.e., $\mathbf{u}^i \oplus \mathbf{u}^j = (\mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}^i) \oplus (\mathbf{t}^j \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}^j) = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1)$. After the base-OTs, P_S holds $G(\mathbf{k}_i^{s_i})$ and $G(\mathbf{k}_j^{s_j})$ and in Step 2a of Protocol 1, P_R computes and sends $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}^i$ and $\mathbf{u}^j = G(\mathbf{k}_j^0) \oplus G(\mathbf{k}_j^1) \oplus \mathbf{r}^j$. Now

PROTOCOL 1 (Passive-secure OT extension protocol of [16])

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \leq j \leq m$.
- **Input of P_R:** m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ and $\ell = \kappa$.
- **Oracles and cryptographic primitives:** The parties use an ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a correlation robust-function $H : [m] \times \{0,1\}^\ell \to \{0,1\}^n$ (see §3.3).

1. *Initial OT Phase:*
   (a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in \{0,1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   (b) The parties invoke the $\ell \times OT_\kappa$-functionality, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

   For every $1 \leq i \leq \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{t}^i$ for $1 \leq i \leq \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \leq j \leq m$.
2. *OT Extension Phase:*
   (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to P_S for every $1 \leq i \leq \ell$.
   (b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
   (c) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
   (d) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where:
   $$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$
   (e) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.
3. **Output:** P_R outputs $(x_1^{r_1}, \ldots, x_m^{r_m})$; P_S has no output.

note that P_S can compute the XOR of the strings he received in the base-OTs $G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j})$ as well as the "inverse" XOR of the strings received in the base-OTs $G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}) = G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j$ if and only if P_R has correctly used $\mathbf{r}^i = \mathbf{r}^j$. However, P_S cannot check whether the "inverse" XOR is correct, since it has no information about $G(\mathbf{k}_i^{\overline{s_i}})$ and $G(\mathbf{k}_j^{\overline{s_j}})$ (this is due to the security of the base-OTs that guarantees that P_S receives the keys $\mathbf{k}_i^{s_i}, \mathbf{k}_i^{s_j}$ only, and learns nothing about $\mathbf{k}_i^{\overline{s_i}}, \mathbf{k}_j^{\overline{s_j}}$). We solve this problem by having P_R commit to the XORs of all strings $h_{i,j}^{p,q} = H(G(\mathbf{k}_i^p) \oplus G(\mathbf{k}_j^q))$, for all combinations of $p, q \in \{0,1\}$. Now, given $h_{i,j}^{s_i,s_j}$, $h_{i,j}^{\overline{s_i},\overline{s_j}}$, P_S checks that $h_{i,j}^{s_i,s_j} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}))$, and that $h_{i,j}^{\overline{s_i},\overline{s_j}} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_i^{s_j}) \oplus \mathbf{u}^i \oplus \mathbf{u}^j) = H(G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}))$. This check passes if $\mathbf{r}^i = \mathbf{r}^j$ and $h_{i,j}^{p,q}$ were set correctly.

If a malicious P_R tries to cheat and has chosen $\mathbf{r}^i \neq \mathbf{r}^j$, it has to convince P_S by computing $h_{i,j}^{p,q} = H(G(\mathbf{k}_i^p) \oplus G(\mathbf{k}_j^q) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$ for all $p, q \in \{0, 1\}$. However, P_S can check the validity of $h_{i,j}^{s_i,s_j} = H(G(\mathbf{k}_i^{s_i}) \oplus G(\mathbf{k}_j^{s_j}))$ while P_R remains oblivious to s_i, s_j. Hence, P_R can only convince P_S by guessing s_i, s_j, computing $h_{i,j}^{s_i,s_j}$ correctly and $h_{i,j}^{\overline{s_i},\overline{s_j}} = H(G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$, which P_R cannot do better than with probability $1/2$. This means that P_R can only successfully learn ρ bits but will be caught except with probability $2^{-\rho}$. The full description of our new protocol is given in Protocol 2. We give some more explanations regarding the possibility of the adversary to cheat during the consistency check in §3.1.

We note that learning few bits of the secret $\mathbf{s}$ does not directly break the security of the protocol once $|\mathbf{s}| > \kappa$. In particular, the values $\{H(\mathbf{t}_j \oplus \mathbf{s})\}_j$ are used to mask the inputs $\{x_j^{1-r_j}\}_j$. Therefore, when H is modelled as a random oracle and enough bits of $\mathbf{s}$ remain hidden from the adversary, each value $H(\mathbf{t}_j \oplus \mathbf{s})$ is random, and the adversary cannot learn the input $x_j^{1-r_j}$. For simplicity we first prove security of our protocol in the random-oracle model. We later show that H can be replaced with a variant of a correlation-robustness assumption.

The advantage of our protocol over [30] is that P_S does not need to reveal any information about s_i, s_j when checking the consistency between r^i and r^j (as long as P_R does not cheat, in which case it risks getting caught). Hence, it can force P_R to check that $\mathbf{r}^i$ equals any $\mathbf{r}^j$, for $1 \leq j \leq \ell$ without disclosing any information.

Section outline. In the following, we describe our basic protocol and prove its security (§3.1). We then show how to reduce the number of consistency checks to achieve better performance (§3.2), and how to replace the random oracle with a weaker correlation robustness assumption (§3.3). Finally, we show how our protocol can be used to achieve covert security (§3.4).

3.1 The Security of Our Protocol

Malicious sender. The original OT extension protocol of [16] already provides security against a malicious P_S. Our checks do not add any capabilities for a malicious sender, since they consist of messages from the receiver to the sender only. Thus, by a simple reduction to the original protocol, one can show that our protocol is secure in the presence of a malicious sender.

Simulating a malicious receiver. In the case of a malicious receiver, the adversary may not use the same $\mathbf{r}$ in the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$, and as a result learn some bits from the secret $\mathbf{s}$. Therefore, we add a consistency check of $\mathbf{r}$ to the semi-honest protocol of [16]. However, this verification of consistency of $\mathbf{r}$ is not perfectly sound, and the verification may still pass even when the receiver sends few $\mathbf{u}$'s that do not define the same $\mathbf{r}$. This makes the analysis a bit more complicated.

For every $1 \leq i \leq \ell$, let $\mathbf{r}^i \overset{\text{def}}{=} \mathbf{u}^i \oplus G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1)$ that is, the "input" $\mathbf{r}^i$ which is implicitly defined by $\mathbf{u}^i$ and the base-OTs.

PROTOCOL 2 (Our active-secure OT extension protocol)

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \leq j \leq m$.
- **Input of P_R:** m selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ and statistical security parameter ρ. It is assumed that $\ell = \kappa + \rho$.
- **Oracles and cryptographic primitives:** The parties use an ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^m$, and random-oracle H (see §3.3 for instantiation of H.)

1. *Initial OT Phase:*
   (a) P_S initializes a random vector $\mathbf{s} = (s_1, \ldots, s_\ell) \in \{0,1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   (b) The parties invoke the $\ell \times OT_\kappa$-functionality, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \leq i \leq \ell$.

   For every $1 \leq i \leq \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{t}^i$ for $1 \leq i \leq \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \leq j \leq m$.
2. *OT Extension Phase (Part I):*
   (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to P_S for every $1 \leq i \leq \ell$.
3. *Consistency Check of* $\mathbf{r}$*:* (the main change from Protocol 1)
   (a) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_R defines the four values:

   $$h_{\alpha,\beta}^{0,0} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0)) \qquad h_{\alpha,\beta}^{0,1} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1)) \ ,$$
   $$h_{\alpha,\beta}^{1,0} = H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0)) \qquad h_{\alpha,\beta}^{1,1} = H(G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^1)) \ .$$

   It then sends $\mathcal{H}_{\alpha,\beta} = (h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}, h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1})$ to P_S.
   (b) For every pair $\alpha, \beta \subseteq [\ell]^2$, P_S knows $s_\alpha, s_\beta, \mathbf{k}_\alpha^{s_\alpha}, \mathbf{k}_\beta^{s_\beta}, \mathbf{u}^\alpha, \mathbf{u}^\beta$ and checks that:
      i. $h_{\alpha,\beta}^{s_\alpha, s_\beta} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}))$.
      ii. $h_{\alpha,\beta}^{\overline{s_\alpha}, \overline{s_\beta}} = H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta)$ $\quad (= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{r}^\alpha \oplus \mathbf{r}^\beta))$.
      iii. $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$.

   In case one of these checks fails, P_S aborts and outputs $\bot$.
4. *OT Extension Phase (Part II):*
   (a) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
   (b) Let $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{q}^i$. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. (Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.)
   (c) P_S sends (y_j^0, y_j^1) for every $1 \leq j \leq m$, where:

   $$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$$

   (d) For $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.
5. **Output:** P_R outputs $(x_1^{r_1}, \ldots, x_m^{r_m})$; P_S has no output.

We now explore how the matrices Q, T are changed when the adversary uses inconsistent $\mathbf{r}$'s. Recall that when the receiver uses the same $\mathbf{r}$, then $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$. However, in case of inconsistent $\mathbf{r}$'s, we get that $\mathbf{q}^i = (s_i \cdot \mathbf{r}^i) \oplus \mathbf{t}^i$. The case of $\mathbf{q}_j$ is rather more involved; let $R = \left[\mathbf{r}^1 \mid \ldots \mid \mathbf{r}^\ell\right]$ denote the $m \times \ell$ matrix where its ith column is $\mathbf{r}^i$, and let $\mathbf{r}_j$ denote the jth row of the matrix R. For two strings of the same length $\mathbf{a} = (a_1, \ldots, a_k), \mathbf{b} = (b_1, \ldots, b_k)$, let $\mathbf{a} * \mathbf{b}$ define the entry-wise product, that is $\mathbf{a} * \mathbf{b} = (a_1 \cdot b_1, \ldots, a_k \cdot b_k)$. We get that $\mathbf{q}_j = (\mathbf{r}_j * \mathbf{s}) \oplus \mathbf{t}_j$ (note that in an honest execution, $\mathbf{r}_j$ is the same bit everywhere). The sender masks the inputs (x_j^0, x_j^1) with $(H(j, \mathbf{q}_j), H(j, \mathbf{q}_j \oplus \mathbf{s}))$.

In order to understand better the value $\mathbf{q}_j$, let $\mathbf{r} = (r_1, \ldots, r_m)$ be the string that occurs the most from the set $\{\mathbf{r}^1, \ldots, \mathbf{r}^\ell\}$, and let $\mathcal{U} \subset [\ell]$ be the set of all indices for which $\mathbf{r}^i = \mathbf{r}$ for all $i \in \mathcal{U}$. Let $B = [\ell] \setminus \mathcal{U}$ be the complementary set, that is, the set of all indices for which for every $i \in B$ it holds that $\mathbf{r}^i \neq \mathbf{r}$. As we will see below, except with some negligible probability, the verification phase guarantees that $|\mathcal{U}| \geq \ell - \rho$. Thus, for every $1 \leq j \leq m$, the vector $\mathbf{r}_j$ (which is the jth row of the matrix R), can be represented as $\mathbf{r}_j = (r_j \cdot \mathbf{1}) \oplus \mathbf{e}_j$, where $\mathbf{1}$ is the all one vector of size ℓ, and $\mathbf{e}_j$ is some error vector with Hamming distance at most ρ from $\mathbf{0}$. Note that the non-zero indices in $\mathbf{e}_j$ are all in B. Thus, we conclude that:

$$\mathbf{q}_j = (\mathbf{s} * \mathbf{r}_j) \oplus \mathbf{t}_j = (\mathbf{s} * (r_j \cdot \mathbf{1} \oplus \mathbf{e}_j)) \oplus \mathbf{t}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j) \ .$$

Recall that in an honest execution $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$, and therefore the only difference is the term $(\mathbf{s} * \mathbf{e}_j)$. Moreover, note that $\mathbf{s} * \mathbf{e}_j$ completely hides all the bits of $\mathbf{s}$ that are in $\mathcal{U}$, and may expose only the bits that are in B. Thus, the consistency check of $\mathbf{r}$ guarantees two important properties: First, that almost all the inputs are consistent with some implicitly defined string $\mathbf{r}$, and thus the bits r_j are uniquely defined. Second, the set of inconsistent inputs (i.e., the set B) is small, and thus the adversary may learn only a limited amount of bits of $\mathbf{s}$.

The consistency checks of r. We now examine what properties are guaranteed by our consistency check, for a single pair (α, β). The malicious receiver P_R first sends the set of keys $\mathcal{K} = \{\mathbf{k}_i^0, \mathbf{k}_i^1\}$ to the base-OT protocol, and then sends all the values $(\mathbf{u}^1, \ldots, \mathbf{u}^\ell)$ and the checks $\mathcal{H} = \{\mathcal{H}_{\alpha,\beta}\}_{\alpha,\beta}$. In the simulation, the simulator can choose $\mathbf{s}$ only after it receives all these messages (this is because the adversary gets no output from the invocation of the OT primitive). Thus, for a given set of messages that the adversary outputs, we can ask what is the number of secrets $\mathbf{s}$ for which the verification will pass, and the number for which it will fail. If the verification passes for some given $\mathcal{T} = (\mathcal{K}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \mathcal{H})$ and some secret $\mathbf{s}$, then we say that $\mathcal{T}$ is consistent with $\mathbf{s}$; In case the verification fails, we say that $\mathcal{T}$ is inconsistent.

The following Lemma considers the values that the adversary has sent regarding some pair (α, β), and considers the relation to the pair of bits (s_α, s_β) of the secret $\mathbf{s}$. We have:

Lemma 31 *Let $\mathcal{T}_{\alpha,\beta} = \{\mathcal{H}_{\alpha,\beta}, \mathbf{u}^\alpha, \mathbf{u}^\beta, \{\mathbf{k}_\alpha^b\}_b, \{\mathbf{k}_\beta^b\}_b\}$ and assume that H is a collision-resistant hash-function. We have:*

1. *If $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathcal{T}_{\alpha,\beta}$ is consistent with (s_α, s_β), then it is inconsistent with $(\overline{s_\alpha}, \overline{s_\beta})$.*
2. *If $\mathbf{r}^\alpha = \mathbf{r}^\beta$ and $\mathcal{T}_{\alpha,\beta}$ is consistent with (s_α, s_β), then it is consistent also with $(\overline{s_\alpha}, \overline{s_\beta})$.*

Proof: For the first item, assume that $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and that $\mathcal{T}_{\alpha,\beta}$ is consistent both with (s_α, s_β) and $(\overline{s_\alpha}, \overline{s_\beta})$. Thus, from the check of consistency of (s_α, s_β):

$$h_{\alpha,\beta}^{s_\alpha,s_\beta} = H\left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta})\right), \qquad h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}} = H\left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta\right) \ ,$$

and that $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$. In addition, from the check of consistency of $(\overline{s_\alpha}, \overline{s_\beta})$ it holds that:

$$h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}} = H\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}})\right), \qquad h_{\alpha,\beta}^{s_\alpha,s_\beta} = H\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta\right) \ ,$$

and that $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$. This implies that:

$$H\left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta})\right) = h_{\alpha,\beta}^{s_\alpha,s_\beta} = H\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta\right) \ ,$$

and from the collision resistance property of H we get that:

$$G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}) = G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta \ .$$

Recall that $\mathbf{r}^\alpha \stackrel{\text{def}}{=} \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$, and $\mathbf{r}^\beta \stackrel{\text{def}}{=} \mathbf{u}^\beta \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$. Combining the above, we get that $\mathbf{r}^\alpha = \mathbf{r}^\beta$, in contradiction.

For the second item, once $\mathbf{r}^\alpha = \mathbf{r}^\beta$, we get that $\mathbf{u}^\alpha \oplus \mathbf{u}^\beta = G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1) \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$ and it is easy to see that if the consistency check of (s_α, s_β) holds, then the consistency check of $(\overline{s_\alpha}, \overline{s_\beta})$ holds also. ∎

Lemma 31 implies what attacks the adversary can do, and what bits of $\mathbf{s}$ it can learn from each such an attack. In the following, we consider a given partial transcript $\mathcal{T}_{\alpha,\beta} = ((\mathbf{k}_\alpha^0, \mathbf{k}_\alpha^1, \mathbf{k}_\beta^0, \mathbf{k}_\beta^1), (\mathbf{u}^\alpha, \mathbf{u}^\beta), \mathcal{H}_{\alpha,\beta})$ and analyze what the messages might be, and what the adversary learns in case the verification passes. Let $\mathbf{r}^\alpha = \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$ and $\mathbf{r}^\beta$ defined analogously. We consider 4 types:

1. **Type 1: correct.** In this case, it holds that $\mathbf{r}^\alpha = \mathbf{r}^\beta$, and for every $(a, b) \in \{0,1\}^2$: $h_{\alpha,\beta}^{a,b} = H\left(G(\mathbf{k}_\alpha^a) \oplus G(\mathbf{k}_\beta^b)\right)$. The verification passes for every possible value of (s_α, s_β).
2. **Type 2: $\mathbf{r}^{\boldsymbol{\alpha}} = \mathbf{r}^{\boldsymbol{\beta}}$, but $\boldsymbol{\mathcal{H}_{\alpha,\beta}}$ is incorrect.** In this case, the adversary sent $\mathbf{u}^\alpha, \mathbf{u}^\beta$ that define the same $\mathbf{r}$. However, it may send hashes $\mathcal{H}_{\alpha,\beta}$ that are incorrect (i.e., for some $(a, b) \in \{0,1\}^2$, it may send: $h_{\alpha,\beta}^{a,b} \neq H(G(\mathbf{k}_\alpha^a) \oplus G(\mathbf{k}_\beta^b)))$. However, from Lemma 31, if $\mathbf{r}^\alpha = \mathbf{r}^\beta$ and $\mathcal{H}_{\alpha,\beta}$ is consistent with (s_α, s_β) then it is also consistent with $(\overline{s_\alpha}, \overline{s_\beta})$.
   Thus, a possible attack of the adversary, for instance, is to send correct hashes for some bits $(0, 0)$ and $(1, 1)$, but incorrect ones for $(0, 1)$ and $(1, 0)$.

The verification will pass with probability 1/2, exactly if (s_α, s_β) are either $(0,0)$ or $(1,1)$, but it will fail in the other two cases (i.e., $(1,0)$ or $(0,1)$). We therefore conclude that the adversary may learn the relation $s_\alpha \oplus s_\beta$, and gets caught with probability 1/2.

3. **Type 3: $\mathbf{r}^{\boldsymbol{\alpha}} \neq \boldsymbol{r}^{\boldsymbol{\beta}}$ and $\boldsymbol{\mathcal{H}_{\alpha,\beta}}$ is incorrect in two positions.** In this case, for instance, the adversary can set the values $h^{0,0}_{\alpha,\beta}, h^{0,1}_{\alpha,\beta}$ correctly (i.e., $h^{0,0}_{\alpha,\beta} = H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^0_\beta))$ and $h^{0,1}_{\alpha,\beta} = H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^1_\beta)))$ and set values $h^{1,0}_{\alpha,\beta}, h^{1,1}_{\alpha,\beta}$, accordingly, such that the verification will pass for the cases of $(s_\alpha, s_\beta) = (0,0)$ or $(0,1)$. That is, it sets:
$$h^{1,0}_{\alpha,\beta} = H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^1_\beta) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta)$$
(and it sets $h^{1,1}_{\alpha,\beta}$ in a similar way). In this case, the adversary succeeds with probability 1/2 and learns that $s_\alpha = 0$ in case the verification passes. Similarly, it can guess the value of s_β and set the values accordingly. For conclusion, the adversary can learn whether its guess was correct, and in which case it learns exactly one of the bits s_α or s_β but does not learn anything about the other bit.
4. **Type 4: $\mathbf{r}^{\boldsymbol{\alpha}} \neq \mathbf{r}^{\boldsymbol{\beta}}$ and $\boldsymbol{\mathcal{H}_{\alpha,\beta}}$ is incorrect in three positions.** In this case, the adversary may guess both bits $(s_\alpha, s_\beta) = (a, b)$ and set $h^{a,b}_{\alpha,\beta}$ correctly, set $h^{\overline{a},\overline{b}}_{\alpha,\beta}$ accordingly (i.e., such that the verification will pass for (a,b)), but will fail for any one of the other cases. In this case, the adversary learns the values (s_α, s_β) entirely, but succeeds with probability of at most 1/4.

Note that whenever $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$, the adversary may pass the verification of the pair (α, β) with probability of at most 1/2. This is because it cannot send consistent hashes for all possible values of (s_α, s_β), and must, in some sense, "guess" either one of the bits, or both (i.e., Type 3 or Type 4). However, an important point that makes the analysis more difficult is the fact that the two checks are not necessarily independent. That is, in case where $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and $\mathbf{r}^\beta \neq \mathbf{r}^\gamma$, although the probability to pass each one of the verification of (α, β) and (β, γ) separately is at most 1/2, the probability to pass both verifications together is higher than 1/4, and these two checks are not independent. This is because the adversary can guess the bit s_β, and set the hashes as in Type 3 in both checks. The adversary will pass these two checks if it guesses s_β correctly, with probability 1/2.

Theorem 32 *Assuming that H is a random oracle, G is a pseudo-random generator, Protocol 2 with $\ell = \kappa + \rho$ securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary, where κ is the symmetric security parameter and ρ is the statistical security parameter.*

Proof Sketch: The simulator $\mathcal{S}$ invokes the malicious receiver and plays the role of the base-OT trusted party and the honest sender. It receives from the adversary its inputs to the base-OTs, and thus knows the values $\{\mathbf{k}^0_i, \mathbf{k}^1_i\}^\ell_{i=1}$. Therefore, it can compute all the values $\mathbf{r}^1, \ldots, \mathbf{r}^\ell$ when it receives the messages

$\mathbf{u}^1, \ldots, \mathbf{u}^\ell$. It computes the set of indices $\mathcal{U}$, and extracts $\mathbf{r}$. It then performs the same checks as an honest sender, in Step 3 of Protocol 2, and aborts the execution if the adversary is caught cheating. Then, it sends the trusted party the value $\mathbf{r}$ that it has extracted, and learns the inputs $x_1^{r_1}, \ldots, x_m^{r_m}$. It computes $\mathbf{q}_j$ as instructed in the protocol (recall that these $\mathbf{q}_j$ may contain the additional "shift" $\mathbf{s} * \mathbf{e}_j$) and use some random values for all $\{y_j^{\overline{r_j}}\}_{j=1}^m$. The full description of the simulator is given in the full proof in the full version [2].

Since the values $\{y_j^{\overline{r_j}}\}_{j=1}^m$ are random in the ideal execution, and equal $\{x_j^{\overline{r_j}} \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})\}$ in the real execution, a distinguisher may distinguish between the real and ideal execution once it makes a query of the form $(j, \mathbf{q}_j \oplus \mathbf{s})$ to the random oracle. We claim, however, that the probability that the distinguisher will make such a query is bounded by $(t+1)/|S|$, where t is the number of queries it makes to the random oracle, and S is the set of all possible secrets $\mathbf{s}$ that are consistent with the view that it receives. Thus, once we show that $|S| > 2^\kappa$, the probability that it will distinguish between the real and ideal execution is negligible in κ.

However, the above description is too simplified. First, if the adversary performs few attacks of Type 2, it learns information regarding $\mathbf{s}$ from the mere fact that the verification has passed. Moreover, recall that $y_j^{r_j} = x_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$, and that the adversary can control the values $\mathbf{t}_j$ and $\mathbf{e}_j$. Recall that $\mathbf{e}_j$ is a vector that is all zero in positions that are in $\mathcal{U}$, and may vary in positions that are in B. This implies that by simple queries to the random oracle, and by choosing the vectors $\mathbf{e}_j$ cleverly, the adversary can totally reveal the bits $\mathbf{s}_B$ quite easily. We therefore have to show that the set B is small, while also showing that the set of consistent secrets is greater than 2^κ (that is, $|S| \geq 2^\kappa$).

Let $\mathcal{T} = \{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^\ell, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \{H_{\alpha,\beta}\}_{\alpha,\beta}\}$, i.e., the values that the adversary gives during the execution of the protocol. Observe that the simulator chooses the secret $\mathbf{s}$ only after $\mathcal{T}$ is determined (since the adversary receives no output from the execution of the base-OT primitive, we can assume that). We divide all possible $\mathcal{T}$ into two sets, $\mathcal{T}_{\mathsf{good}}$ and $\mathcal{T}_{\mathsf{bad}}$, defined as follows:

$$\mathcal{T}_{\mathsf{good}} = \left\{\mathcal{T} \mid \Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] > 2^{-\rho}\right\} \text{ and}$$
$$\mathcal{T}_{\mathsf{bad}} = \left\{\mathcal{T} \mid \Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] \leq 2^{-\rho}\right\},$$

where $\mathsf{consistent}(\mathcal{T}, \mathbf{s})$ is a predicate that gets 1 when the verification passes for the transcript $\mathcal{T}$ and the secret $\mathbf{s}$, and 0 otherwise. The probability is taken over the choice of $\mathbf{s}$. For a given $\mathcal{T}$, let $\mathcal{S}(\mathcal{T})$ be the set of all possible secrets $\mathbf{s} \in \{0,1\}^\ell$, that are consistent with $\mathcal{T}$. That is: $\mathcal{S}(\mathcal{T}) = \{\mathbf{s} \in \{0,1\}^\ell \mid \mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\}$. Therefore, it holds that: $\Pr_{\mathbf{s}}\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] = \frac{|\mathcal{S}(\mathcal{T})|}{2^\ell}$, and thus $|\mathcal{S}(\mathcal{T})| = 2^\ell \cdot \Pr\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right]$. As a result, for every $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, it holds that $|\mathcal{S}(\mathcal{T})| > 2^\ell \cdot 2^{-\rho} = 2^{\ell-\rho}$. This already guarantees that once the adversary sends transcript $\mathcal{T}$ that will pass the verification with high probability, then the number of possible secrets that are consistent with this transcript is quite large, and therefore it is hard to guess the exact secret $\mathbf{s}$ that was chosen.

We claim that if $|\mathcal{U}| \leq \ell - \rho$ (i.e., $|B| > \rho$), then it must hold that $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$ and the adversary gets caught with high probability. Intuitively, this is because we have ρ independent checks, $\{(u, b)\}$, where $u \in \mathcal{U}$ and $b \in B$, which are pairwise disjoint. As we saw, here we do have independency between the checks, and the adversary can pass this verification with probability at most $2^{-\rho}$. Thus, once the adversary outputs transcript $\mathcal{T}$ for which $|B| > \rho$, we have that $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$.

We conclude that if the adversary outputs $\mathcal{T}$ for which $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$ then it gets caught both in the ideal and the real execution, and the simulation is identical. When $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, we get that the number of possible secrets is greater than $2^{\ell-\rho}$, and in addition, $|B| < \rho$. This already gives us a proof for the case where $\ell = \kappa + 2\rho$: Even if we give the distinguisher all the bits $\mathbf{s}_B$ (additonal ρ bits), the set of all possible secrets that are consistent with $\mathcal{T}$ is of size $2^{\ell-2\rho} \geq 2^{\kappa}$.

In the full proof in the full version [2], we show that $\ell = \kappa + \rho$ base-OTs are sufficient. In particular, we show that for a given transcript $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ the bits $\mathbf{s}_B$ are *exactly the same* for all the secrets that are consistent with $\mathcal{T}$. As a result, the at most ρ bits $\mathbf{s}_B$ that we give to the distinguisher do not give it any new information, and we can set $\ell = \kappa + \rho$. ■

3.2 Reducing the Number of Checks

In Protocol 2, in the consistency check of $\mathbf{r}$, we check all possible pairs $(\alpha, \beta) \in [\ell]^2$. In order to achieve higher efficiency, we want to reduce the number of checks.

Let $G = (V, E)$ be a graph for which $V = [\ell]$, and an edge (α, β) represents a check between $\mathbf{r}^{\alpha}$ and $\mathbf{r}^{\beta}$. In Protocol 2 the receiver asks for all possible edges in the graph (all pairs). In order to achieve better performance, we would like to reduce the number of pairs that we check. In particular, the protocol is changed so that in Step 3 of Protocol 2 the sender chooses some set of pairs (edges) $E' \subseteq V^2$, and the receiver must respond with the quadruples $\mathcal{H}_{\alpha,\beta}$ for every $(\alpha, \beta) \in E'$ that it has been asked for. The sender continues with the protocol only if all the checks have passed successfully.

Observe that after sending the values $\mathbf{u}^1, \ldots, \mathbf{u}^{\ell}$, the sets $\mathcal{U}$ and B (which are both subsets of $[\ell]$) are implicitly defined. In case that the set B is too large, we want to catch the adversary cheating with probability of at least $1 - 2^{-\rho}$. In order to achieve this, we should have ρ edges between B and $\mathcal{U}$ that are pairwise non-adjacent. That is, in case the adversary defines B that is "too large", we want to choose a set of edges E' that contains a matching between B and $\mathcal{U}$ of size of at least ρ.

Note, however, that the sender chooses the edges E' with no knowledge whatsoever regarding the identities of $\mathcal{U}$ and B, and thus it needs to choose a graph such that (with overwhelming probability), for any possible choice of a large B, there will be a ρ-matching between B and $\mathcal{U}$.

In protocol 3 we modify the consistency check of $\mathbf{r}$ that appears in Step 3 of Protocol 2. The sender chooses for each vertex $\alpha \in [\ell]$ exactly μ out-neighbours uniformly at random. We later show that with high probability the set E' that is chosen contains a ρ-matching between the two sets B and $\mathcal{U}$, even for a very small value of μ (for instance, $\mu = 3$ or even $\mu = 2$).

PROTOCOL 3 (Modification for Protocol 2, Fewer Checks)
The parties run Protocol 2 with the following modifications:
Step 3 – *Consistency Check of* $\mathbf{r}$*:* (modified)

1. P_S chooses μ functions $\phi_0, \ldots, \phi_{\mu-1}$ uniformly at random, where $\phi_i : [\ell] \to [\ell]$. It sends $\phi_0, \ldots, \phi_{\mu-1}$ to the receiver P_R.
2. For every pair $\alpha \in [\ell], i \in [\mu]$, let $(\alpha, \beta) = (\alpha, \phi_i(\alpha))$. P_R defines the four values:

$$h^{0,0}_{\alpha,\beta} = H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^0_\beta)) \qquad h^{0,1}_{\alpha,\beta} = H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^1_\beta)) \ ,$$
$$h^{1,0}_{\alpha,\beta} = H(G(\mathbf{k}^1_\alpha) \oplus G(\mathbf{k}^0_\beta)) \qquad h^{1,1}_{\alpha,\beta} = H(G(\mathbf{k}^1_\alpha) \oplus G(\mathbf{k}^1_\beta)) \ .$$

   It then sends $\mathcal{H}_{\alpha,\beta} = (h^{0,0}_{\alpha,\beta}, h^{0,1}_{\alpha,\beta}, h^{1,0}_{\alpha,\beta}, h^{1,1}_{\alpha,\beta})$ to P_S.
3. P_S checks that it receives $H_{\alpha,\phi_i(\alpha)}$ for every $\alpha \in [\ell]$ and $i \in [\mu]$. Then, for each pair $(\alpha, \beta) = (\alpha, \phi(\alpha))$ it checks that:
   (a) $h^{s_\alpha,s_\beta}_{\alpha,\beta} = H(G(\mathbf{k}^{s_\alpha}_\alpha) \oplus G(\mathbf{k}^{s_\beta}_\beta))$.
   (b) $h^{\overline{s_\alpha},\overline{s_\beta}}_{\alpha,\beta} = H(G(\mathbf{k}^{s_\alpha}_\alpha) \oplus G(\mathbf{k}^{s_\beta}_\beta) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta) \quad (= H(G(\mathbf{k}^{\overline{s_\alpha}}_\alpha) \oplus G(\mathbf{k}^{\overline{s_\beta}}_\beta) \oplus \mathbf{r}^\alpha \oplus \mathbf{r}^\beta))$.
   (c) $\mathbf{u}^\alpha \neq \mathbf{u}^\beta$.
   In case one of these checks fails, P_S aborts and outputs $\perp$.

In our modified protocol, the adversary again first outputs $\mathcal{T} = \{\{\mathbf{k}^0_i, \mathbf{k}^1_i\}^\ell_{i=1}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell\}$. Then, the set of checks $\Phi = \{\phi_0, \ldots, \phi_{\mu-1}\}$ is chosen, and the adversary responds with $\mathcal{H} = \{\{\mathcal{H}_{\alpha,\phi_i(\alpha)}\}_{\alpha,\phi_i}\}$. We can assume that the actual secret $\mathbf{s}$ is chosen only after $\mathcal{T}, \Phi$ and $\mathcal{H}$ are determined. Similarly to the proof of Theorem 32, for a given transcript $(\mathcal{T}, \Phi, \mathcal{H})$ and a secret $\mathbf{s}$, we define the predicate $\mathsf{consistent}((\mathcal{T}, \Phi, \mathcal{H}), \mathbf{s})$ that gets 1 if and only if the verification is passed for the secret $\mathbf{s}$ (that is, that the sender does not output $\perp$). For a given $\mathcal{T}$ and set of checks Φ, let $\mathcal{H}_{\mathcal{T},\Phi}$ be the set of responds that maximizes the probability to pass the verification, that is:

$$\mathcal{H}_{\mathcal{T},\Phi} \stackrel{\text{def}}{=} \mathrm{argmax}_{\mathcal{H}}\{\Pr\left[\mathsf{consistent}_{\mathbf{s}}((\mathcal{T}, \Phi, \mathcal{H}), \mathbf{s}) = 1\right]\} \ .$$

We separate all possible transcripts $(\mathcal{T}, \Phi)$ to two sets $\mathcal{T}_{\mathsf{good}}$ and $\mathcal{T}_{\mathsf{bad}}$ such that:

$$\mathcal{T}_{\mathsf{good}} = \{(\mathcal{T}, \Phi) \mid \Pr_{\mathbf{s}}\left[\mathsf{consistent}((\mathcal{T}, \Phi, \mathcal{H}_{\mathcal{T},\Phi}), \mathbf{s}) = 1\right] > 2^{-\rho}\} \text{ and}$$
$$\mathcal{T}_{\mathsf{bad}} = \{(\mathcal{T}, \Phi) \mid \Pr_{\mathbf{s}}\left[\mathsf{consistent}((\mathcal{T}, \Phi, \mathcal{H}_{\mathcal{T},\Phi}), \mathbf{s}) = 1\right] \leq 2^{-\rho}\} \ .$$

Observe that if a pair $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{bad}}$, then no matter what set $\mathcal{H}$ the adversary sends, it gets caught with probability of at least $1 - 2^{-\rho}$.

The following claim bounds the size of the set B. It states that if the adversary $\mathcal{A}$ outputs $\mathcal{T}$ that defines $|\mathcal{U}| < \kappa$, then with probability $1 - 2^{-\rho}$ the sender will choose Φ such that $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{bad}}$.

Claim 33 *Let $\mathcal{T}$ be as above, and let $\mathcal{U}$ be the largest set of indices such that for every $\alpha, \beta \in \mathcal{U}$, $\mathbf{r}^\alpha = \mathbf{r}^\beta$. Assume that $|\mathcal{U}| < \kappa$. Then, for appropriate choice of parameters $|B|, \mu$, it holds that:*

$$\Pr_{\Phi}[(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{bad}}] \geq 1 - 2^{-\rho}.$$

Proof: The partial transcript $\mathcal{T}$ defines the two sets B and $\mathcal{U}$. Viewing the base-OTs $[\ell]$ as vertices in a graph, and the pairs of elements that are being checked as edges $E' = \{(\alpha, \phi_i(\alpha)) \mid \alpha \in [\ell], i \in [\mu]\}$, we have a bipartite graph $(B \cup U, E')$ where each vertex has at least μ out edges. We want to show that with probability $1-2^{-\rho}$ (over the choice of Φ), there exists a ρ-matching between $\mathcal{U}$ and B. Once there is a ρ-matching, the adversary passes the verification phase with probability of at most $2^{-\rho}$, and thus the pair $(\mathcal{T}, \Phi)$ is in $\mathcal{T}_{\mathsf{bad}}$.

In order to show that in a graph there is a ρ-matching between B and $\mathcal{U}$, we state the following theorem which is a refinement of Hall's well-known theorem (see [27]). Let $N_{\mathcal{U}}(S)$ denote the set of neighbours in $\mathcal{U}$, for some set of vertices $S \subseteq B$, that is, $N_{\mathcal{U}}(S) = \{u \in \mathcal{U} \mid \exists v \in S, \ s.t. \ (u, v) \in E'\}$. We have:

Theorem 34 *There exists a matching of size ρ between B and $\mathcal{U}$ if and only if, for any set $S \subseteq B$, $|N_{\mathcal{U}}(S)| \geq |S| - |B| + \rho$.*

Note that we need to consider only subsets $S \subseteq B$ for which $|S| \geq |B| - \rho$ (otherwise, the condition holds trivially).

The choice of Φ is equivalent to choosing μ out edges for each vertex uniformly. We will show that for every subset of $S \subseteq B$ with $|S| \geq |B| - \rho$, it holds that $|N_{\mathcal{U}}(S)| \geq |S| - |B| + \rho$.

Let $S \subseteq B$ and $T \subset \mathcal{U}$. Let $X_{S,T}$ be an indicator random variable for the event that all the out-edges from S go to $B \cup T$, and all the out-edges of $\mathcal{U} \setminus T$ do not go to S (we use the term "out edges" even though the graph is not directed; our intention is simply the edges connecting these parts). As a result, $|N_{\mathcal{U}}(S)| \leq |T|$. Then, the probability that $X_{S,T}$ equals 1 is the probability that all the $\mu \cdot |S|$ out edges of S go to $B \cup T$ only, and all the $\mu \cdot (|\mathcal{U}| - |T|)$ out edges of $\mathcal{U} \setminus T$ go to $\{\ell\} \setminus S$ only. Since we have independency everywhere, we have:

$$\Pr[X_{S,T} = 1] = \left(\frac{|B| + |T|}{\ell}\right)^{|S| \cdot \mu} \cdot \left(\frac{\ell - |S|}{\ell}\right)^{(|\mathcal{U}| - |T|) \cdot \mu}$$

We are interested in the event $\sum X_{S,T}$ for all $S \subseteq B, T \subseteq \mathcal{U}$ $s.t.$ $|B| - \rho \leq |S| \leq |B|, |T| \leq |S| - |B| + \rho$ (denote this condition by $(\star)$), and we want to show that it is greater than 0 with very low probability. We have:

$$\Pr\left[\sum_{S,T,\ \text{s.t.}\ (\star)} X_{S,T} > 0\right] \leq \sum_{S,T\ \text{s.t.}\ (\star)} \Pr\left[X_{S,T}=1\right] \tag{1}$$

$$\leq \sum_{S,T\ \text{s.t.}\ (\star)} \left(\frac{|B|+|T|}{\ell}\right)^{|S|\cdot\mu} \cdot \left(\frac{\ell-|S|}{\ell}\right)^{(|\mathcal{U}|-|T|)\cdot\mu} \tag{2}$$

$$= \sum_{|S|=|B|-\rho}^{|B|} \sum_{|T|=0}^{|S|-|B|+\rho} \binom{|B|}{|S|}\cdot\binom{|\mathcal{U}|}{|T|}\cdot\left(\frac{|B|+|T|}{\ell}\right)^{|S|\cdot\mu} \cdot \left(\frac{\ell-|S|}{\ell}\right)^{(|\mathcal{U}|-|T|)\cdot\mu}$$

We do not provide an asymptotic analysis for this expression since we loose accuracy by using any upper bound for any one of the terms in it. We next compute this expression for some concrete choice of parameters. We note that the use of the union bound in Eq. (2) already reduces the tightness of our analysis, which may cause more redundant checks or base-OTs than actually needed.

■

Concrete choice of parameters. Claim 33 states that the bound is achieved for an appropriate choice of parameters. We numerically computed the probability in Eq. (1) for a variety of parameters, and obtained that the probability is less than $2^{-\rho}$ with $\rho = 40$, for the following parameters:

$\boldsymbol{\kappa}$	128							80			
$\lvert\boldsymbol{B}\rvert$	62	49	46	44	43	42	41	53	48	46	42
$\boldsymbol{\mu}$	2	3	4	5	6	8	15	3	4	5	10
$\boldsymbol{\ell}$	190	177	174	172	171	170	169	133	128	125	122
#-checks	380	531	696	860	1,026	1,360	2,535	399	512	625	1,220

In Section 4.2, we run empirical tests to see which parameters perform best in which setting. We recall that in case we check all pairs (i.e., Protocol 2), we have either $\ell = \kappa + \rho = 128 + 40 = 168$ base-OTs with $\binom{\ell}{2} = 14{,}028$ checks, or $\ell = \kappa + \rho = 80 + 40 = 120$ base-OTs with 7,140 checks.

3.3 Correlation Robustness Instead of a Random Oracle

In this section, we show how a correlation robustness assumption (with respect to a high min-entropy source) suffices for proving the security of our protocol.

Correlation robust function. We first recall the standard definition of a correlation robust function from [16], as well as a stronger version of the assumption. Let U_ℓ denote the uniform distribution over strings of length ℓ.

Definition 35 (Correlation Robustness) *An efficiently computable function $H : \{0,1\}^\kappa \to \{0,1\}^n$ is* correlation robust *if it holds that:*

$$\{\mathbf{t}_1, \ldots, \mathbf{t}_m, H(\mathbf{t}_1 \oplus \mathbf{s}), \ldots, H(\mathbf{t}_m \oplus \mathbf{s})\} \overset{c}{\equiv} \{U_{m\cdot\kappa+m\cdot n}\}$$

where $\mathbf{t}_1, \ldots, \mathbf{t}_m, \mathbf{s} \in \{0,1\}^\kappa$ *are uniformly and independently distributed. H is* strongly correlation robust *if for every* $\mathbf{t}_1, \ldots, \mathbf{t}_m \in \{0,1\}^\kappa$ *it holds that:*

$$\{H(\mathbf{t}_1 \oplus \mathbf{s}), \ldots, H(\mathbf{t}_m \oplus \mathbf{s})\} \overset{c}{\equiv} \{U_{m \cdot n}\}$$

where $\mathbf{s} \in \{0,1\}^\kappa$ *is uniform.*

Another way of looking at this is as a type of *pseudorandom function*. Specifically, define $F_{\mathbf{s}}(\mathbf{t}) = H(\mathbf{t} \oplus \mathbf{s})$. Then, H is correlation robust if and only if F is a weak pseudorandom function, and H is strongly correlation robust if and only if F is a (non-adaptive) pseudorandom function. For proving the security of our protocol, we need to consider the above notions but where $\mathbf{s}$ is chosen from a high min-entropy source. Thus, we consider the case where H is also somewhat an extractor.

Let X be a random variable taking values from $\{0,1\}^\ell$. The min-entropy of $\mathcal{X}$, denoted $H_\infty(\mathcal{X})$, is: $H_\infty(\mathcal{X}) \overset{\mathrm{def}}{=} \min_x \left\{\log \frac{1}{\Pr[\mathcal{X}=x]}\right\} = -\log\left(\max_x \{\Pr[\mathcal{X} = x]\}\right)$. If a source $\mathcal{X}$ has a min entropy κ we say that $\mathcal{X}$ is a "κ-source". For instance, a κ-source may be κ uniform and independent bits, together with some $\ell - \kappa$ fixed bits (in an arbitrary order), or κ uniform bits with some $\ell - \kappa$ bits that dependent arbitrarily on the first random bits. We are now ready to define min-entropy correlation robustness.

Definition 36 (Min-Entropy Correlation Robustness) *An efficiently computable function* $H : \{0,1\}^\ell \to \{0,1\}^n$ *is* κ-min-entropy correlation robust *if for all (efficiently samplable) κ-sources $\mathcal{X}$ on $\{0,1\}^\ell$ it holds that:*

$$\{\mathbf{t}_1, \ldots, \mathbf{t}_m, H(\mathbf{t}_1 \oplus \mathbf{s}), \ldots, H(\mathbf{t}_m \oplus \mathbf{s})\} \overset{c}{\equiv} \{U_{m \cdot \ell + m \cdot n}\}$$

where $\mathbf{t}_1, \ldots, \mathbf{t}_m$ *are chosen uniformly and independently at random from* $\{0,1\}^\ell$*, and* $\mathbf{s} \leftarrow \mathcal{X}$*. H is* κ-min-entropy strongly correlation robust *if for all (efficiently samplable) κ-sources $\mathcal{X}$ on $\{0,1\}^\ell$ and every (distinct)* $\mathbf{t}_1, \ldots, \mathbf{t}_m \in \{0,1\}^\ell$ *it holds that:*

$$\{H(\mathbf{t}_1 \oplus \mathbf{s}), \ldots, H(\mathbf{t}_m \oplus \mathbf{s})\} \overset{c}{\equiv} \{U_{m \cdot n}\}$$

where $\mathbf{s} \leftarrow \mathcal{X}$*.*

In Protocol 2, the values that are used to mask the inputs of the sender are $H(\mathbf{t}_j), H(\mathbf{t}_j \oplus \mathbf{s})$ (or, $H(\mathbf{t}_j \oplus (s * \mathbf{e}_j)), H(\mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j) \oplus \mathbf{s})$ in case the adversary uses different $\mathbf{r}^i$'s). Since the receiver is the one that effectively chooses the $\mathbf{t}_j$'s values, it may choose values that are not distributed uniformly or even choose them maliciously. As a result, we prove the security of Protocol 2 in its current form using the *strong* κ-min-entropy correlation robustness assumption.

However, it is also possible to modify the protocol and rely only on κ-min-entropy correlation robustness, as follows. In Step 4c (of Protocol 2), in each iteration $1 \le j \le m$, the sender chooses a random value $\mathbf{d}_j \in \{0,1\}^\ell$, and sends the values $(\mathbf{d}_j, y_j^0, y_j^1)$, where:

$$y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j \oplus \mathbf{d}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{d}_j \oplus \mathbf{s}) \ .$$

Then, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus \mathbf{d}_j)$. Since the $\mathbf{d}_j$ values are chosen last, this ensures that the values used inside H are always uniformly distributed. Thus, κ-min-entropy correlation robustness suffices.

In Step 3 of Protocol 2 we also use the function H; however, the property that is needed from H for these invocations is collision resistance and not correlation robustness. Therefore, to explicitly emphasize the differences between the two assumptions, we say that the parties use a collision resistant function h in Step 3 of the protocol, and a (variant of) correlation robust function in Step 4c.

Theorem 37

1. *Assume that H is strongly κ-min-entropy correlation robust, h is a collision resistant function and G is a pseudo-random generator. Then, Protocol 2 securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*
2. *Assume that H is κ-min-entropy correlation robust, h is a collision resistant function and G is a pseudo-random generator. Then, the above-described modified protocol securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

A proof for this theorem appears in the full version [2].

3.4 Achieving Covert Security

In this section, we present a more efficient protocol (with fewer base-OTs and checks) with the property that any deviation from the protocol that can result in a breach of security will be detected with probability at least 1/2. For details on the definition of covert security, we refer to [3]. Our protocol below is secure under the *strong explicit-cheat formulation* with deterrent factor $\epsilon = \frac{1}{2}$.

As in the malicious case, given the set of keys $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}$, and the messages $u^1, \ldots, u^\ell$, the sets B and $\mathcal{U}$ are implicitly defined, and we want to catch the adversary if its behavior defines a set B with "high" cardinality. Here, in contrast to the malicious case, we will be content with catching the adversary with probability 1/2, instead of $1 - 2^{-\rho}$ as in the case of malicious adversaries. As we will show below, our approach for the consistency check of $\mathbf{r}$ enables us to achieve a deterrent factor of 1/2 at the cost of very few consistency checks. Concretely, it will be enough to use 7 checks of pairs only.

The protocol. In Step 3 of Protocol 2, the sender chooses t random pairs $\{(\alpha_i, \beta_i)\}_{i=1}^t$ uniformly and independently at random, and sends them to the receiver. The receiver sends $\mathcal{H}_{\alpha_i,\beta_i}$ for each pair (α_i, β_i) that it was asked. Then, the sender performs the same checks as in the previous protocol: It checks that the receiver replied with hashes for all the pairs (α_i, β_i) that it was asked for, and that the hashes that were sent are correct (i.e., as in Step 3b of Protocol 2).

The analysis. Although at first sight the analysis below ignores attacks of Type 2, these attacks are still taken into consideration. This is because whenever the adversary tries to cheat and learn bits of $\mathbf{s}$ where $\mathbf{r}^\alpha = \mathbf{r}^\beta$, it gets caught doing so with probability 1/2, which is exactly the deterrent factor. The analysis therefore focuses on the case that the adversary cheats when $|B|$ is "too large", and shows that when we have t checks and $|B|$ is large enough, then the probability that the adversary passes the verification is less than 1/2.

We again consider the graph of checks, and let $V = [\ell]$ and the edges are all possible checks. We divide $[\ell]$ to B and $\mathcal{U}$, and we show that when using t checks, the probability that the adversary succeeds to pass the verification when B is "large" is less than 1/2.

There are ℓ^2 edges overall, where $2|B| \cdot |\mathcal{U}|$ are edges between B and $\mathcal{U}$, and $|B|^2 + |\mathcal{U}|^2$ edges are between B and B, or $\mathcal{U}$ and $\mathcal{U}$. We say that an edge is "good" if it goes between B and $\mathcal{U}$. Recall that in such a check, the adversary is caught with probability at least 1/2.

For the first edge that is chosen, the probability that it is a good edge is $2|B| \cdot |\mathcal{U}|/\ell^2$. However, once this specific edge between B and $\mathcal{U}$ is chosen, an edge between B and $\mathcal{U}$ that is pairwise non-adjacent with the previously chosen edge is not longer good, since the probability that the adversary will get caught here is not 1/2. Therefore, we denote by good_i the probability of choosing the $(i+1)$th "good" edge. That is, the probability that edge e_j is good, conditioned on the event that i good edges were previously chosen in the set $\{e_1, \ldots, e_{j-1}\}$. We have that:

$$\mathsf{good}_i = \frac{2 \cdot (|B| - i) \cdot (|\mathcal{U}| - i)}{\ell^2}.$$

This holds because once a good edge is chosen, we do not want to choose an edge that is adjacent to it. As a result, with each good edge that is chosen, the effective size of the set B and $\mathcal{U}$ is decreased by 1.

In contrast, we denote by bad_i the probability that the next chosen edge is bad, given that there were i previous good edges. That is, a bad edge is either an edge between B and B, an edge between $\mathcal{U}$ and $\mathcal{U}$, or is adjacent to one of the $2i$ vertices of the previously chosen good edges. This probability is as follows:

$$\mathsf{bad}_i = \frac{|B|^2 + |\mathcal{U}|^2 + 2i \cdot |\mathcal{U}| + 2i \cdot |B| - 2i^2}{\ell^2} = \frac{|B|^2 + |\mathcal{U}|^2 + 2i(\ell - i)}{\ell^2}$$

That is, a bad edge can be either an edge from B to B, $\mathcal{U}$ to $\mathcal{U}$, or an edge between the i vertices that were chosen with any other vertex. Note, however, that there are some edges that are counted twice and thus we remove $2i^2$. In addition, observe that $\mathsf{good}_i + \mathsf{bad}_i = 1$.

When we have t checks, we may have between 0 to t good edges. In case there are d good edges, the probability that the adversary succeeds to cheat is 2^{-d}. In order to ease the calculation, let good be the maximal probability of $\mathsf{good}_0, \ldots, \mathsf{good}_{t-1}$, and let bad be the maximal probability of $\mathsf{bad}_0, \ldots, \mathsf{bad}_t$. We get that:

$$\mathsf{good} = \frac{2 \cdot |B| \cdot |\mathcal{U}|}{\ell^2}$$

and for $t < \ell/2$:

$$\mathsf{bad} = \frac{|B|^2 + |\mathcal{U}|^2 + 2t(\ell - t)}{\ell^2}\ .$$

Now, consider the edges $e_1, \ldots, e_t$. The probability that the adversary succeeds in its cheating is the union of succeeds in cheating in each possible combination of checks. In particular, we may have $d = 0, \ldots, t$ good edges, and for each d, there are $\binom{t}{d}$ possible ways to order d good edges and $t-d$ "bad" edges. Finally, when we have d good edges, the probability that the adversary succeeds to cheat is 2^{-d}. We therefore have that the probability that the adversary successfully cheats without being caught is less than:

$$\sum_{d=0}^{t} \binom{t}{d} \cdot \mathsf{good}^d \cdot \mathsf{bad}^{t-d} \cdot 2^{-d} = \sum_{d=0}^{t} \binom{t}{d} \cdot \left(\frac{1}{2} \cdot \mathsf{good}\right)^d \cdot \mathsf{bad}^{t-d} = \left(\frac{1}{2} \cdot \mathsf{good} + \mathsf{bad}\right)^t .$$

It is easy to verify that this probability is less than 0.5 for $|B| = 38$, $|\mathcal{U}| = 128$ (and so overall $\ell = 166$), with only 7 checks. In which case, we have that $\mathsf{good} = 0.353$, $\mathsf{bad} = 0.728$, and the probability is less than 0.495.

4 Performance Evaluation

We experimentally compare the performance of our protocols to previous works using the same programming language and running benchmarks on the same machines: We first describe our implementation (§4.1), empirically evaluate and compare the identified active and covert parameters of §3.2 and §3.4 (§4.2), and compare our work to the active-secure protocol of [30] with optimizations of [11] and to the passive-secure protocol of [16] with optimizations from [1] (§4.3).

Benchmarking Environment: We run our experiments in two settings: a local setting and a cloud setting. In the *local setting*, the sender and receiver routines run on two Desktop PCs which each have 16 GB RAM, an Intel Haswell i7-4770K CPU with 4 cores and AES-NI support, and are connected via Gigabit Ethernet. In the *cloud setting*, we run the OT sender routine on an Amazon EC2 m3.medium instance with a 2.5 GHz, Intel Xeon E5-2670v2 CPU and 3.75 memory located in North Virginia (US East) and run the OT receiver routine on one of our Desktop PCs in Europe. The average bandwidth usage in the cloud setting was 52 MBit/s and the average ping latency (round-trip-time) was 95 ms.

4.1 Implementation

We build on the passive-secure and publicly available OT extension C++ implementation of [1]. We perform the OT extension protocol and consistency checks block-wise, i.e., we split m OTs into b blocks of size $w = 2^{18}$, with $b = \lceil \frac{m}{w} \rceil$. These blocks can be processed independently of each other and using multiple threads. For all experiments we evaluate the random OT version of [1], since the additional overhead to obtain the traditional OT functionality is equal for all protocols, and output $n =$ 8-bit strings. For the base-OTs we use [28] for the passive-secure OT extension protocol and [31] in decryption mode with security based on the Decisional Diffie-Helmann (DDH) assumption for the covert- and active-secure OT extension protocols; we implement both using elliptic curves. We assume $\kappa = 128$-bit long-term security with $\rho = 40$ statistical security. Further implementation details are given in Appendix §A.

4.2 Parameter Evaluation

We evaluate the asymptotic communication and run-time in the local and cloud setting on 2^{23} random OTs for our most promising active security (cf. Table 3.2) and covert security (cf. §3.4) parameters, and compare them to the active-secure protocol of [30] with $\ell = \lceil \frac{8}{3}\kappa \rceil = 342$ base-OTs and $\ell/2 = 171$ checks, and to the passive-secure protocol of [16] with $\ell = 128$ base-OTs and no checks. The results are depicted in Table 2 where the parameters are given as (#base-OTs;#checks). We also include the pairwise comparison Protocol 2 (which performs all possible checks) with parameters (168;14,028) and discuss its special features in Appendix §A.3.

Security	**Active**						**Covert**	**Passive**
Parameters	[30]	190;380	177;531	174;696	170;1,360	168;14,028	166;7	[16]
Comm. [MB]	342	191	178	175	**173**	195	**166**	**128**
Local Setting								
Run-time [s]	16.988	**11.938**	13.201	18.218	25.918	221.382	**10.675**	**9.579**
Cloud Setting								
Run-time [s]	110.223	64.698	63.845	**63.712**	83.414	454.595	**46.718**	**33.838**

Table 2. Run-time and communication for active, covert, and passive security using different parameters (#base-OTs;#checks) on 2^{23} random OTs. Minimum values are marked in bold.

For the communication we can observe that our parameter sets have $50\% - 55\%$ of the communication of [30]. Furthermore, while decreasing the number of base-OTs reduces the overall communication until 170 base-OTs, the overhead in communication for sending the consistency check hashes outweighs the gains from the reduced number of base-OTs. Hence, using less than 170 base-OTs for block-size $w = 2^{18}$ would increase both communication and computation complexity.

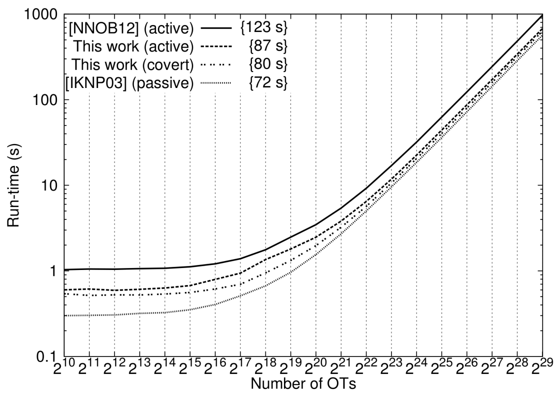
For the run-time we can observe that our best-performing parameter has 70% of the run-time of [30] in the local setting and 58% of the run-time in the cloud setting. Furthermore, the best-performing parameter differs between the local and cloud setting: while the (190;380) parameter performs best in the local setting, the (174;696) parameter achieves the lowest run-time in the cloud setting. This can be explained by the smaller bandwidth of the cloud setting, which influences the run-time of all parameters differently. For instance, when switching from the local to the cloud setting, the run-time of [30] increases by factor 6.5, whereas that of our pairwise comparison Protocol 2 with parameter (168;14,028) only increases by factor 2. As expected, the covert parameter (166;7) performs better than the parameters for active security.

4.3 Comparison with Related Work

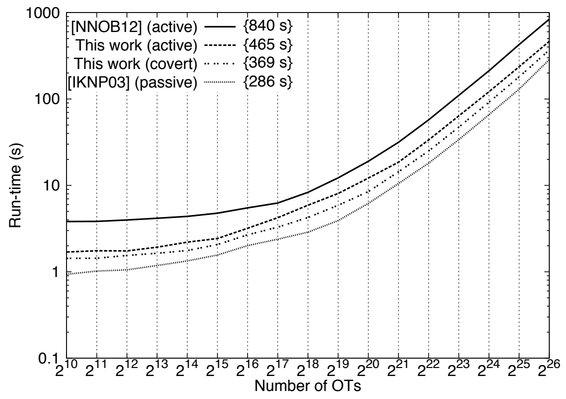
We empirically evaluate and compare our protocol on a varing number of OTs in its active and covert versions to the passive-secure OT extension protocol of [16] with optimizations of [18, 1], and the active-secure OT extension protocol of [30] with optimizations of [11]. The results for the local and cloud setting are given in Figure 1. We benchmark the protocols on an exponentially increasing number of OTs: from 2^{10} to 2^{29} for the local setting and from 2^{10} to 2^{26} for the cloud setting. The passive-secure [16] serves as bottom-line for the performance of the other protocols to show the (small) gap to the covert- and active-secure protocols. For our protocol we use the parameters from our parameter evaluation in §4.2 which were shown to perform best in the respective setting, i.e., (190;380) for the local setting, (174;696) for the cloud setting, and (166;7) for covert security. For the [30] protocol we use $\ell = \lceil \frac{8}{3}\kappa \rceil = 342$ base-OTs and $\ell/2 = 171$ checks. We excluded the active-secure protocol of [20], since its communication overhead is at least two orders of magnitude higher than for the evaluated protocols and simply transferring the required data would result in higher run-times than those of the other protocols.

For the results in the local setting we can observe that our active-secure OT extension protocol outperforms the [30] protocol for all OTs tested on and scales better with increasing number of OTs. Furthermore, our active-secure protocol converges towards the passive-secure [16] protocol when more OTs are performed, decreasing the overhead for active security down to 121% for 2^{26} OTs, compared to an overhead of 171% for the [30] protocol. The convergence of our protocol can be explained by the amortizing costs of the consistency checks. Since the consistency checks are performed on blocks of fixed width 2^{18}, their amortization happens for a larger number of OTs. The covert version of our protocol has only 111% overhead compared to the passive-secure protocol.

In the cloud setting, the performance of all protocols decreases, as expected. However, the performance of the passive-secure protocol decreases less significantly compared to the covert- and active-secure protocols. This can be explained by the smaller communication complexity of the passive-secure protocol, since the run-time overhead scales with the communication overhead of the respective protocol. For the active-secure protocol of [30] with communication overhead of 267% compared to the passive-secure protocol, the run-time overhead increases from 171% to 294%. In comparison, for our active-secure protocol with communication overhead of 136%, the run-time overhead increases from 121% to 163%. Finally, for our covert protocol with communication overhead of 129%, the run-time overhead increases from 111% to 129%.



(a) Local Setting



(b) Cloud Setting

Fig. 1. Run-time for random OT extension protocols for 8-bit strings with active, covert, and passive security in the local- and cloud setting. Time for 2^{26} OTs given in {}.

References

1. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM Computer and Communications Security (CCS'13). pp. 535–548. ACM (2013), code: `http://encrypto.de/code/OTExtension`
2. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries (full version). IACR Cryptology ePrint Archive 2015, 061 (2015), online: `http://eprint.iacr.org/2015/061`
3. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. Journal of Cryptology 23(2), 281–343 (2010)
4. Beaver, D.: Correlated pseudorandomness and the complexity of private computations. In: Symposium on the Theory of Computing (STOC'96). pp. 479–488. ACM (1996)

5. Damgård, I., Lauritsen, R., Toft, T.: An empirical study and some improvements of the MiniMac protocol for secure computation. In: Security and Cryptography for Networks (SCN'14). LNCS, vol. 8642, pp. 398–415. Springer (2014)
6. Damgård, I., Zakarias, S.: Constant-overhead secure computation of Boolean circuits using preprocessing. In: Theory of Cryptography Conference (TCC'13). LNCS, vol. 7785, pp. 621–641. Springer (2013)
7. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: An efficient and scalable protocol. In: ACM Computer and Communications Security (CCS'13). pp. 789–800. ACM (2013)
8. Ejgenberg, Y., Farbstein, M., Levy, M., Lindell, Y.: SCAPI: the secure computation application programming interface. IACR Cryptology ePrint Archive 2012, 629 (2012), http://eprint.iacr.org/2012/629
9. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Communications of the ACM 28(6), 637–647 (1985)
10. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B.: Faster maliciously secure two-party computation using the GPU. In: Security and Cryptography for Networks (SCN'14). LNCS, vol. 8642, pp. 358–379. Springer (2014)
11. Frederiksen, T.K., Nielsen, J.B.: Fast and maliciously secure two-party computation using the GPU. In: Applied Cryptography and Network Security (ACNS'13). LNCS, vol. 7954, pp. 339–356. Springer (2013)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Symposium on Theory of Computing (STOC'87). pp. 218–229. ACM (1987)
13. Harnik, D., Ishai, Y., Kushilevitz, E., Nielsen, J.B.: OT-combiners via secure computation. In: Theory of Cryptography Conference (TCC'08). LNCS, vol. 4948, pp. 393–411. Springer (2008)
14. Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Advances in Cryptology – CRYPTO'14. LNCS, vol. 8617, pp. 458–475. Springer (2014)
15. Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: Advances in Cryptology (CRYPTO'88). LNCS, vol. 403, pp. 8–26. Springer (1988)
16. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Advances in Cryptology – CRYPTO'03. LNCS, vol. 2729, pp. 145–161. Springer (2003)
17. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently. In: ACM Computer and Communications Security (CCS'13). pp. 955–966. ACM (2013)
18. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Advances in Cryptology – CRYPTO'13. LNCS, vol. 8043, pp. 54–70. Springer (2013)
19. Kreuter, B., Shelat, A., Shen, C.: Billion-gate secure computation with malicious adversaries. In: USENIX Security Symposium'12. pp. 285–300. USENIX (2012)
20. Larraia, E.: Extending oblivious transfer efficiently, or - how to get active security with constant cryptographic overhead. In: Progress in Cryptology – LATINCRYPT'14. LNCS, Springer (2014), to appear. Online: http://eprint.iacr.org/2014/692
21. Larraia, E., Orsini, E., Smart, N.P.: Dishonest majority multi-party computation for binary circuits. In: Advances in Cryptology – CRYPTO'14. LNCS, vol. 8617, pp. 495–512. Springer (2014)

22. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Advances in Cryptology – EUROCRYPT'07. LNCS, vol. 4515, pp. 52–78. Springer (2007)

23. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Theory of Cryptography Conference (TCC'11). LNCS, vol. 6597, pp. 329–346. Springer (2011)

24. Lindell, Y., Pinkas, B., Smart, N.P.: Implementing two-party computation efficiently with security against malicious adversaries. In: Security and Cryptography for Networks (SCN'08). LNCS, vol. 5229, pp. 2–20. Springer (2008)

25. Lindell, Y., Riva, B.: Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In: Advances in Cryptology – CRYPTO'14. LNCS, vol. 8617, pp. 476–494. Springer (2014)

26. Lindell, Y., Zarosim, H.: On the feasibility of extending oblivious transfer. In: Theory of Cryptography Conference (TCC'13). LNCS, vol. 7785, pp. 519–538. Springer (2013)

27. Lovász, L., Plummer, M.: Matching Theory. Akadémiai Kiadó, Budapest (1986), also published as Vol. 121 of the North-Holland Mathematics Studies, North-Holland Publishing, Amsterdam

28. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Symposium on Discrete Algorithms (SODA'01). pp. 448–457. ACM/SIAM (2001)

29. Nielsen, J.B.: Extending oblivious transfers efficiently - how to get robustness almost for free. IACR Cryptology ePrint Archive 2007, 215 (2007), online: http://eprint.iacr.org/2007/215

30. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Advances in Cryptology – CRYPTO'12. LNCS, vol. 7417, pp. 681–700. Springer (2012)

31. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Advances in Cryptology – CRYPTO'08. LNCS, vol. 5157, pp. 554–571. Springer (2008)

32. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Advances in Cryptology – ASIACRYPT'09. LNCS, vol. 5912, pp. 250–267. Springer (2009)

33. Rabin, M.O.: How to exchange secrets with oblivious transfer, TR-81 edn. (1981), Aiken Computation Lab, Harvard University

34. Shelat, A., Shen, C.H.: Fast two-party secure computation with minimal assumptions. In: ACM Computer and Communications Security (CCS'13). pp. 523–534. ACM (2013)

35. Yao, A.C.: How to generate and exchange secrets. In: Foundations of Computer Science (FOCS'86). pp. 162–167. IEEE (1986)

A Implementation Details

In this section we provide details about the architecture of our implementation (§A.1), the method we use to allow block-wise evaluation of our protocol and [30] (§A.2), and discuss the benefits of the pairwise-comparison method described in Protocol 2 (§A.3).

A.1 Architecture

We designed the architecture of the active-secure OT extension implementations such that the communication-intensive passive-secure OT extension routine and the computation-intensive checks on receiver side are performed by separate threads and can be further parallelized independently of each other. This architecture allows us to instantiate the implementation specifically to the available resources of the deployment scenario. More detailed, we can perform the communication-intensive operations with as many threads as required to fully utilize the bandwidth and can then focus the remaining processing power on the computationally-intensive operations. This kind of parallelization offers benefits especially for deployment scenarios of OT extension with small bandwidth, where the network is the bottle-neck for OT extension and where further parallelization of communication-intensive operations would only result in congestion on the network interface. Although this architecture favors our protocol which is computationally more intensive than the protocols of [16] and [30], we argue that it nicely fits to today's increasing number of CPU cores.

A.2 3-Step OT Extension

Note that in order to allow block-wise evaluation of our protocol and [30], the base-OTs have to be renewed. For the block-wise evaluation of $m \times OT_n$ in b blocks of width w bits ($b = \lceil \frac{m}{w} \rceil$), we perform a 3-step OT extension: In the first step, we perform $\ell \times OT_{b\ell}$ base-OTs using the protocol of [31]. In the second step, we extend $\ell \times OT_{b\ell}$ to $b\ell \times OT_w$ using the respective active secure OT extension protocol. In the third step, we again perform the OT extension step b-times on each ℓ-bit interval, i.e., we extend $\ell \times OT_w$ to $w \times OT_n$ b-times and thereby obtain $bw \geq m$ OTs on n-bit strings.

A.3 Advantages of the Pairwise Comparison Protocol

Although the pairwise comparison Protocol 2 with parameter (168;14,028) is the slowest in our evaluation in §4.2, we stress that it has several advantages which make it favorable in settings with high computation power. The main advantage is that the receiver can pre-compute all checks directly after the base-OTs, since all combinations are checked and hence the sender does not need to send a mapping to the receiver. Additionally, if a computationally powerful device such as a GPU is present, the receiver can use it for computing the checks in parallel.

B Active Secure OT Extension of [30]

In Protocol 4 we depict the actively-secure OT extension protocol of [30] with optimizations from [11].

PROTOCOL 4 (Active secure OT extension protocol of [30])

- **Input of P_S:** m pairs (x_j^0, x_j^1) of n-bit strings, $1 \le j \le m$.
- **Input of P_R:** m selection bits $\mathbf{r} = (r_1, \dots, r_m)$.
- **Common Input:** Symmetric security parameter κ and $\ell = \lceil \frac{8}{3}\kappa \rceil$.
- **Oracles and primitives:** Ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator G, correlation-robust function H, and random-oracle H'.

1. *Initial OT Phase:*
   (a) P_S initializes a random vector $\mathbf{s} = (s_1, \dots, s_\ell) \in \{0,1\}^\ell$ and P_R chooses ℓ pairs of seeds $\mathbf{k}_i^0, \mathbf{k}_i^1$ each of size κ.
   (b) The parties invoke the $\ell \times OT_\kappa$-functionality, where P_S acts as the *receiver* with input $\mathbf{s}$ and P_R acts as the *sender* with inputs $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ for every $1 \le i \le \ell$.

   For every $1 \le i \le \ell$, let $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Let $T = [\mathbf{t}^1 | \dots | \mathbf{t}^\ell]$ denote the $m \times \ell$ bit matrix where its ith column is $\mathbf{t}^i$ for $1 \le i \le \ell$. Let $\mathbf{t}_j$ denote the jth row of T for $1 \le j \le m$.
2. *OT Extension Phase:*
   (a) P_R computes $\mathbf{t}^i = G(\mathbf{k}_i^0)$ and $\mathbf{u}^i = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to P_S for every $1 \le i \le \ell$.
   (b) For every $1 \le i \le \ell$, P_S defines $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
3. *Consistency Check of* $\mathbf{r}$*:*
   (a) P_S chooses a uniform random permutation $\pi : \{1, ..., \ell\} \mapsto \{1, ..., \ell\}$ with $\pi(\pi(i)) = i$ and sends π to Bob. Let $\Pi(\pi) = \{i | i \le \pi(i)\}$.
   (b) For all $i \in \Pi(\pi)$, P_S computes $d_i = s_i \oplus s_{\pi(i)}$ and $\mathbf{z}^i = \mathbf{q}^i \oplus \mathbf{q}^{\pi(i)}$ sends d_i to P_R.
   (c) P_R computes $\mathbf{z}'^i = (d_i \cdot \mathbf{r}) \oplus \mathbf{t}^i \oplus \mathbf{t}^{\pi(i)}$.
   (d) P_S and P_R check equality between $\mathbf{Z} = \mathbf{z}_1 || ... || \mathbf{z}_{\lfloor \ell/2 \rfloor}$ and $\mathbf{Z}' = \mathbf{z}'_1 || ... || \mathbf{z}_{\lfloor \ell/2 \rfloor}$ as follows:
      i. P_S samples $w \in_R \{0,1\}^\kappa$, computes $\mathbf{c} = H'(\mathbf{Z} || \mathbf{w})$, sends $\mathbf{c}$ to P_R.
      ii. P_R then sends $\mathbf{Z}'$ to P_S.
      iii. P_S checks $\mathbf{Z} \overset{?}{=} \mathbf{Z}'$ and aborts on failure. Else sends $(\mathbf{Z}, \mathbf{w})$ to P_R.
      iv. P_R checks that $\mathbf{Z} \overset{?}{=} \mathbf{Z}'$ and $c \overset{?}{=} H'(\mathbf{Z}' || \mathbf{w})$ and aborts on failure.
   (e) For all $\lfloor \ell/2 \rfloor$ indices in $i \in \Pi(\pi)$ where i is the kth index with $1 \le k \le \lfloor \ell/2 \rfloor$, P_S sets $\mathbf{q}'_k = \mathbf{q}_i$ and $s'_k = s_i$ and P_R sets $\mathbf{t}'_k = \mathbf{t}_i$.
4. *OT Extension (continue):*
   (a) Let $Q' = [\mathbf{q}'^1 | \dots | \mathbf{q}'^{\lfloor \ell/2 \rfloor}]$ denote the $m \times \lfloor \ell/2 \rfloor$ bit matrix where its ith column is $\mathbf{q}'^i$. Let $\mathbf{q}'_j$ denote the jth row of the matrix Q'. (Note that $\mathbf{q}'^i = (s'_i \cdot \mathbf{r}) \oplus \mathbf{t}'^i$ and $\mathbf{q}'_j = (r_j \cdot \mathbf{s}') \oplus \mathbf{t}'_j$.)
   (b) P_S sends (y_j^0, y_j^1) for every $1 \le j \le m$, where $y_j^0 = x_j^0 \oplus H(j, \mathbf{q}'_j)$ and $y_j^1 = x_j^1 \oplus H(j, \mathbf{q}'_j \oplus \mathbf{s}')$.
   (c) For $1 \le j \le m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}'_j)$.
5. **Output:** P_R outputs $(x_1^{r_1}, \dots, x_m^{r_m})$; P_S has no output.