More Efficient Oblivious Transfer Extensions[*]

Gilad Asharov[†] Yehuda Lindell[‡] Thomas Schneider[§] Michael Zohner[§]

December 5, 2017

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 malicious adversaries, that is secure in a random oracle model.

In this work we improve OT extensions with respect to communication complexity, computation complexity, and scalability in the semi-honest, covert, and malicious model. Furthermore, we show how to modify our maliciously secure OT extension protocol to achieve security with respect to a version of correlation robustness instead of the random oracle. We also provide specific optimizations of OT extensions that are tailored to the use of OT in various secure computation protocols such as Yao's garbled circuits and the protocol of Goldreich-Micali-Wigderson, which reduce the communication complexity even further. We experimentally verify the efficiency gains of our protocols and optimizations.

Note (December 5, 2017): This version includes an important fix of the protocol for the case of a corrupted malicious sender.

Keywords: Oblivious transfer extensions, concrete efficiency, secure computation

1 Introduction

In the setting of secure two-party computation, two parties P_0 and P_1 with respective inputs x and y wish to compute a joint function f on their inputs without revealing anything but the

[*]This paper is a combined and extended version of [ALSZ13] (ACM CCS 2013) and [ALSZ15] (Eurocrypt 2015).

[†]IBM T.J. Watson Research Center, NY. Some of the work was done while at the School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel, and while at the Department of Computer Science, Bar-Ilan University, Israel. `gsasharo@us.ibm.com`

[‡]Department of Computer Science, Bar-Ilan University, Israel. `lindell@biu.ac.il`

[§]Department of Computer Science, TU Darmstadt, Darmstadt, Germany. `{thomas.schneider,michael.zohner}@crisp-da.de` output $f(x, y)$. This captures a large variety of tasks, including privacy-preserving data mining, anonymous transactions, private database search, and many more.

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. Within this goal, optimizing the efficiency of protocols in the semi-honest model serves as an important stepping stone. In our paper, we optimize protocols in the semi-honest model and show how to achieve covert and malicious security at low additional cost.

Practical secure computation. Secure computation has been studied since the mid 1980s, when powerful feasibility results demonstrated that any efficient function can be computed securely [Yao86, GMW87]. However, until recently, the bulk of research on secure computation was theoretical in nature. Indeed, many held the opinion that secure computation will never be practical since carrying out cryptographic operations for every gate in a circuit computing the function (which is the way many protocols work) will never be fast enough to be of use. Due to many works that pushed secure computation further towards practical applications, e.g., [MOR03, MNPS04, FAZ05, BNP08, EFG$^+$09, HEKM11, Mal11, Ker11, SK11, HFKV12, CHK$^+$12, NNOB12, NWI$^+$13, BHKR13, DZ13, KSS13, DLT14, LOS14, DSZ15, FKOS15, BLN$^+$15, LR15], this conjecture has proven to be wrong and it is possible to carry out secure computation of complex functions at speeds that five years ago would have been unconceivable, both in the semi-honest model as well as in the malicious model. For example, in [KSS13] it was shown that a single AES evaluation can be securely computed in 12 ms even with security against malicious adversaries. This has applications to private database search and also to mitigating server breaches in the cloud by sharing the decryption key for sensitive data between two servers and never revealing it (thereby forcing an attacker to compromise the security of two servers instead of one). In addition, several applications have a circuit size of several million up to billions of AND gates, which would have until recently been thought impossible to evaluate securely. For instance, the Edit-Distance circuit of [HEKM11] has a size of *1.29 billion AND gates*. Hence, the underlying cryptographic operations that are performed in secure computation protocols need to efficiently process large-scale circuits.

Oblivious transfer and extensions. In an *oblivious transfer (OT)* [Rab81, EGL85], a sender with a pair of input strings (x_0, x_1) interacts with a receiver who inputs a choice bit r. The result is that the receiver learns x_r without learning anything about x_{1-r}, while the sender learns nothing about r. Oblivious transfer is an extremely powerful tool and the foundation for almost all efficient protocols for secure computation. Notably, Yao's garbled-circuits protocol [Yao86] requires OT for every input bit of one party, and the GMW protocol [GMW87] requires OT for every AND gate of the circuit. Accordingly, the efficient instantiation of OT is of crucial importance as is evident in many recent works that focus on efficiency, e.g., [MNPS04, HKS$^+$10, HCE11, HEKM11, HMEK11, HEK12, KSS12, NNOB12, GKK$^+$12, CHK$^+$12, NNOB12, SZ13, LOS14, PSZ14, DLT14, DSZ15, FKOS15, BLN$^+$15, LR15]. The best known OT protocol in the semi-honest and malicious case is that of [CO15], which achieves around 10,000 1-out-of-2 OTs per second using one thread. However, if millions or even billions of oblivious transfers need to be carried out, this will become prohibitively expensive. We give concrete examples for typical applications requiring a large number of OTs next:

Example 1.1 *The AES circuit has* $\sim$ 10,000 *AND gates (cf. [NNOB12]) and requires* 20,000 *passively secure OTs when evaluated with GMW and* $\sim 2^{20}$ *actively secure OTs when evaluated with TinyOT (*≥ 40 *OTs (aBits) per AND gate [LOS14]).*

Example 1.2 *The PSI circuit (Sort-Compare-Shuffle) of [HEK12] has* $\mathsf{O}(bn \log n)$ *AND gates and for* $n = 65{,}536$ *elements with* $b = 32$*-bits the circuit has* 2^{25} *AND gates and requires* 2^{26} *passively secure OTs when evaluated with GMW and* $\sim 2^{30}$ *actively secure OTs when evaluated with TinyOT.*

Example 1.3 *The PSI protocol of [DCW13] needs* $1.44kn$ *OTs for both, the passively- and actively secure versions of the protocol. For* $n = 1{,}000{,}000$ *elements and security parameter* $k = 128$*, this amounts to* $\sim 2^{27}$ *OTs (*~ 180 *OTs per element).*

To meet this large-scale demand of OTs, *OT extensions* [Bea96, IKNP03] can be used. An OT extension protocol works by running a small number of base-OTs (say, 80 or 128) that are used as a base for obtaining many OTs via the use of cheap symmetric cryptographic operations only. This is conceptually similar to hybrid encryption where instead of encrypting a large message using RSA, which would be too expensive, only a single RSA computation is carried out to encrypt a symmetric key and then the long message is encrypted using symmetric operations only. Such an OT extension can actually be achieved with extraordinary efficiency; specifically, the protocol of [IKNP03] requires only three hash function computations on a single block per oblivious transfer (beyond the initial base-OTs). For active adversaries, OT extensions are somewhat more expensive. Prior to this work, the best known protocol for OT extensions with security against active adversaries was introduced by [NNOB12], which added an overhead of approximately $\frac{8}{3}$ (= 266%) to the passively secure OT extension protocol of [IKNP03].

1.1 Our Contributions and Outline

In this paper, we present more efficient protocols for OT extensions in the semi-honest, covert, and malicious model. Our improvements in the semi-honest model (§4) seem somewhat surprising since the protocol of [IKNP03] sounds optimal given that only three hash function computations are needed per transfer. Interestingly, our protocols do not lower the number of hash function operations. However, we observe that significant cost is incurred due to other factors than the hash function operations. We propose several optimizations that improve computation and communication and outline how to parallelize the semi-honest OT extension. We build on the efficiency improvements of the semi-honest OT extension protocol of [IKNP03] and outline how to extend the protocol to covert and malicious adversaries at a lower cost than the previously best malicious secure OT extension protocol of [NNOB12] (§5). In short, our protocol improves the overhead that comes with extending the passively secure OT extension protocol of [IKNP03] to malicious adversaries from 266% to 150%. Finally, we outline different OT flavors that are specifically designed to be used in secure computation protocols and which reduce the communication and computation even further (§6). We apply our optimizations to the OT extension implementation of [SZ13] (which is based on [CHK$^+$12]) and demonstrate the improvements by extensive experiments (§7).[1] After presenting related work in §3 and preliminaries in §2 our paper is structured as follows:

Faster semi-honest OT extensions §4. We present an improved version of the original OT extension protocol of [IKNP03] with reduced communication and computation complexity. Furthermore, we demonstrate how the OT extension protocol can be processed in independent blocks, allowing OT extension to be parallelized and yielding a much faster runtime (§4.1). In addition, we show how to implement the matrix transpose operation using a cache-efficient algorithm that operates on multiple entries at once (§4.2); this significantly reduces the run-time of the protocol to 41% as can be seen in the LAN experiments in Table 1. Finally, we show how to reduce the communication from the receiver to the sender to 50% (§4.3). This is of great importance since local computations of the OT extension protocol are so fast that the communication is often the bottleneck, especially when running the protocol over the Internet or even wireless networks (cf. WAN results in Table 1 and Figure 2).

Faster covert and malicious OT extensions §5. We present our improved malicious OT extension protocol which improves on the previously best malicious OT extension protocol of [NNOB12]. We first present the basic protocol (§5.1) and prove its security (§5.2). The basic protocol adds very low communication overhead to the semi-honest version but incurs a high computation overhead. We show how to reduce the computation at the cost of increased communication, which results in better overall efficiency (§5.3). The resulting protocol decreases the communication overhead for obtaining actively secure OT extension from 266% for [NNOB12] to 150%. We then outline how to modify the protocol to replace the random oracle with a weaker correlation robustness assumption (§5.4). Finally, we show how to modify our protocol to achieve covert security (§5.5).

Extended OT functionality §6. Our improved protocols can be used in any setting that regular OT can be used. However, with a mind on the application of secure computation, we further optimize the protocol by taking into account its use in secure computation in §6. We outline four OT flavors that are specifically designed to be used in secure computation protocols and which reduce the communication and computation even further: *Correlated OT*, *Sender Random OT*, *Receiver Random OT*, and *Random OT*. Correlated OT (C-OT, §6.1) is suitable for secure computation protocols that require varying correlated inputs, such as Yao's garbled circuits protocol with the free-XOR technique [Yao86, KS08] or the arithmetic multiplication routine of [DSZ15]. Sender Random OT (SR-OT, §6.2) and Receiver Random OT (RR-OT, §6.3) are suitable where the input of the sender (or receiver) can be random but the input of the receiver (sender) needs to be chosen. Finally, Random OT (R-OT §6.4) is a combination of Sender Random and Receiver Random OT and can be used where the inputs of sender and receiver can be random, such as GMW with multiplication triples [GMW87, SZ13] (cf. §2.7). In all cases, the communication from the sender to the receiver is reduced to 50% (or even less) of the original protocol of [IKNP03].

Experimental evaluation §7. We experimentally verify the performance improvements of our proposed optimizations for OT extension and special purpose OT functionalities in a LAN and a WAN setting. A summary of our results for 2^{24} random OT extensions on 1-bit strings using 4 threads is given in Table 1. Overall, our optimizations improve the run-time and communication of the passively secure OT extension protocol of [IKNP03] by factor 2-3 and 2, respectively, and the run-time and communication for actively secure OT extension by factor 1.3-1.7 and 1.7, respectively.

[1]Our implementation is available online at `http://encrypto.de/code/OTExtension`.

Prot.	Comm. [MB]	Run-Time [s] LAN	WAN	Base-OTs	Security
Passive					
[IKNP03]	508	9.2	39.9	128	CRF
[KK13]	154	7.8	20.8	256	RO
This (§4)	254	3.8	18.8	128	CRF
Covert					
This (§5.5)	330	4.5	26.1	166	CRF / RO
Active					
[Lar14]	196,688*	-	-	323	CRF
[NNOB12]	682	9.1	50.4	342	RO
This (§5)	378	7.3	30.5	190	CRF / RO
[KOS15]	256*	-	-	128	RO

Table 1: Empirical communication and run-time for 2^{24} random OT extensions on 1-bit strings with $\kappa = 128$ bit security evaluated using 4 threads in a LAN and WAN setting (cf. §7). The security assumption is given as correlation robust function assumption (CRF) or random oracle assumption (RO) cf. §2.2. Numbers with * are estimated.

1.2 Concurrent and Independent Related Work

Parallel to and independently of our work on passively secure OT extension, [KK13] introduced an efficient 1-out-of-N OT extension protocol and outlined the same optimization for reducing the communication from the receiver to the sender by 50% that we propose in §4.3. When transferring short strings, their 1-out-of-N OT extension protocol can be broken down into $\log_2(N)$ 1-out-of-2 OTs that require less communication than $\log_2(N)$ executions of our 1-out-of-2 OT extension protocol (cf. Table 1). We implement and compare their protocol on 1-out-of-2 OT on 1 bit in §7.4.

Most recently, a new actively secure OT extension protocol has been introduced in [KOS15] which works in the random oracle model and achieves nearly the same communication and computation overhead as the passively secure protocol of [IKNP03]. Their protocol is conceptually similar to ours (and to that of [NNOB12]) but performs the checks on the base-OTs in parallel instead of checking individual pairs. Furthermore, their check routine can be implemented very efficiently using the AES new instructions (AES-NI), resulting in very little computational overhead over the passively secure variant. The authors prove that, if one uses κ base-OTs, the protocol provides $2^{\kappa-c}$ computational security against a malicious receiver who is able to learn c bits with probability at most 2^{-c} where κ is the computational security parameter. In contrast to the work of [KOS15], we prove that our protocol is secure in the weaker, min-entropy correlation-robust model (cf. §5.4).

1.3 Extensions over Previous Work

This work combines and extends our works previously published at ACM CCS 2013 [ALSZ13] and Eurocrypt 2015 [ALSZ15]. We have made the following improvements over these versions:

- §5: Detailed proof of the malicious OT extension and parameter estimation (§5.2).
- §6: Extended special purpose OT functionalities in particular Receiver Random OT for GMW (§6.3) and formal proofs of security.

- §7: Extended experiments, in particular comparison with the passively-secure 1-out-of-N OT extension of [KK13] and using parallelism for actively secure OT extension (§7.4), and the k-min entropy correlation (§7.5).

2 Preliminaries

In the following we give preliminaries for our paper. We describe our security paramters (§2.1) and definitions (§2.2) and give an overview of oblivious transfer (§2.3), oblivious transfer extensions (§2.4), Yao's garbled circuits protocol (§2.5), the GMW protocol of Goldreich-Micali-Wigderson (§2.6), and outline how to evaluate AND gates in GMW using oblivious transfer (§2.7).

2.1 Security Parameters

Our protocols use 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 poly(κ), except with probability $\mu(\kappa) + 2^{-\rho}$ (a formal definition follows and is based on [LP11]). In our experiments we set $\kappa = 128$ and $\rho = 40$, which is considered to be secure beyond 2030.[2] Table 2 lists usage times (time frames) for different values of the symmetric security parameter κ (*SYM*) and corresponding field sizes for elliptic curve cryptography (ECC) as recommended by NIST [NIS12]. For ECC we use Koblitz curves which had the best performance in our experiments (cf. [EFLL12]).

Security (Time Frames)	**SYM**	**ECC**
Short (legacy)	80	K-163
Medium (< 2030)	112	K-243
Long (> 2030)	128	K-283

Table 2: Security parameters and recommended key sizes.

2.2 Definitions

We let κ denote the security parameter and let ρ denote the statistical security parameter. For a set A, $x \in_R A$ denotes that the element x is chosen uniformly at random from A. We first define indistinguishability respectively to both security and statistical security parameter, as in [LP11]. A distribution ensemble $X = \{X(a, \kappa, \rho)\}_{\kappa,\rho\in\mathbb{N},a\in\{0,1\}^\kappa}$ is an infinite sequence of random variables. Two distribution ensembles X, Y are (κ, ρ)-*computationally indistinguishable*, denoted $X \overset{\kappa,\rho}{\equiv} Y$ if there exists a constant $0 < c \leq 1$ such that for every nonuniform polynomial time distinguisher D, every $\rho \in \mathbb{N}$, every polynomial $p(\cdot)$ and all large enough $\kappa \in \mathbb{N}$ it holds that for every $a \in \{0,1\}^*$:

$$|\Pr[D(X(a,\kappa,\rho),a,\kappa,\rho)=1)] - \Pr[D(Y(a,\kappa,\rho),a,\kappa,\rho)]| < \frac{1}{p(\kappa)} + \frac{1}{2^{c\cdot\rho}} \tag{1}$$

In protocols where we do not use a statistical security parameter (as the semi-honest protocols in this paper), we use the standard computational indistinguishability definition, which is a special

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

case of the definition above. Specifically, the ensembles X and Y are indexed by a and κ only, and we omit the quantification over ρ and the term $\frac{1}{2^{c\cdot\rho}}$ in Eq. (1). We denote this (standard) indistinguishability by $X \overset{\text{c}}{\equiv} Y$.

Correlation Robust Function. We recall the standard definition of a correlation robust function from [IKNP03], as well as a stronger version of the assumption. Let U_ℓ denote the uniform distribution over strings of length ℓ.

Definition 2.1 (Correlation Robustness) *An efficiently computable function $H : \{0,1\}^\kappa \to \{0,1\}^n$ is* correlation robust *if it holds that:*

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

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:*

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

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

Secure Two-Party Computation. We refer the reader to [Gol04, Chap. 7] and [Can00] for the definitions of security for two-party computation in the presence of semi-honest and malicious adversaries. In the semi-honest we require the standard computational-indistinguishability. For the malicious case, we require (κ, ρ)-indistinguishability between the ideal and the real distributions, rather than just regular computational indistinguishability. We also consider the model of covert adversaries, and refer the reader to [AL10] for appropriate definitions.

2.3 Oblivious Transfer

Oblivious transfer (OT) was first introduced by Rabin [Rab81] 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 re-defined to the 1-out-of-2 OT functionality more commonly used today by [EGL85], where the 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 inputs m pairs of n-bit strings (x_j^0, x_j^1) for $1 \le j \le m$ and the receiver inputs 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 $m \times OT_n$ and call the sender P_S or P_0 and the receiver P_R or P_1.

Several protocols for OT based on different cryptographic assumptions and attacker models were introduced. Most notable are the passively secure OT protocol of [NP01] and the actively secure OT protocols of [PVW08] and [CO15], which are among the most efficient today. However, the impossibility result of [IR88] showed that OT protocols require costly asymmetric cryptography, which greatly limits their efficiency.

2.4 OT Extension

In his seminal work, Beaver [Bea96] introduced *OT extension* protocols, which extend few costly *base-OTs* using symmetric cryptography only. While the first construction of [Bea96] was inefficient and mostly of theoretical interest, the protocol of [IKNP03] showed that OT can be extended efficiently and with very little overhead. We give the semi-honest secure OT extension protocol of [IKNP03] in Protocol 1.

PROTOCOL 1 (Semi-honest secure OT extension protocol of [IKNP03])

- **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 number of base-OTs $\ell = \kappa$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the $\ell \times OT_\kappa$ functionality and use a 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 Def. 2.1).

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 $\ell \times OT_\kappa$-oracle, 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 $1 \leq i \leq \ell$.

   Let $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ be a random $m \times \ell$ bit matrix that is generated by P_R 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 $\mathbf{u}^{(i,0)} = \mathbf{t}^i \oplus G(\mathbf{k}_i^0)$ and $\mathbf{u}^{(i,1)} = \mathbf{t}^i \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $(\mathbf{u}^{i,0}, \mathbf{u}^{i,1})$ to P_S for every $1 \leq i \leq \ell$.
   (b) For every $1 \leq i \leq \ell$, P_S defines $\mathbf{q}^i = \mathbf{u}^{(i,s_i)} \oplus G(\mathbf{k}_i^{s_i})$. (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 $(x_1, \ldots, x_m)$; P_S has no output.

2.5 Yao's Garbled Circuits Protocol

Yao's garbled circuits protocol [Yao86] allows two parties to securely compute an arbitrary function that is represented as Boolean circuit. The sender P_S encrypts the Boolean gates of the circuit using symmetric keys and sends the encrypted function together with the keys that correspond to his input bits to the receiver P_R. P_R then uses a $m \times OT_\kappa$ to obliviously obtain the keys that correspond to his m input bits and evaluates the encrypted function by decrypting it gate by gate. To obtain the output, P_R sends the resulting output keys to P_S or P_S provides a mapping from keys to output bits.

2.6 The GMW Protocol

The protocol of Goldreich, Micali, and Wigderson (GMW) [GMW87] also represents the function to be computed as a Boolean circuit. Both parties secret-share their inputs using the XOR operation and evaluate the Boolean circuit as follows. An XOR gate is computed by locally XORing the shares while an AND gate is evaluated interactively with the help of a multiplication triple [Bea91] which can be precomputed by two random 1-out-of-2 OTs on bits (cf. §2.7). To reconstruct the outputs, the parties exchange their output shares. The performance of GMW depends on the number of OTs and on the depth of the evaluated circuit, since the evaluation of AND gates requires interaction.

2.7 GMW with Random 1-out-of-2 OTs

An AND gate in the GMW protocol can be computed efficiently using the multiplication triple functionality [Bea91], denoted as f^{mult}:

$$f^{mult}(\lambda, \lambda) = ((a_0, b_0, c_0), (a_1, b_1, c_1)) \in_R \{0,1\}^6 \text{ s.t. } c_0 \oplus c_1 = (a_0 \oplus a_1)(b_0 \oplus b_1) \ ,$$

where λ denotes the empty string.

In order to precompute the multiplication triples, previous works suggest to use 1-out-of-4 bit OT [CHK+12, SZ13]. In the following, we present a different approach for generating multiplication triples using two random 1-out-of-2 OTs on bits (R-OT). The R-OT functionality is exactly the same as OT, except that the sender gets two random messages (x_0, x_1) and the receiver gets a random choice-bit a and x_a as *output*. Later in §6.4, we will show that R-OT can be extended more efficiently than OT. In comparison to 1-out-of-4 bit OTs, using two R-OTs only slightly increases the computation complexity (one additional evaluation of G and H and two additional matrix transpositions), but reduces the communication complexity by a factor of 2. Alternatively one could use the 1-out-of-N OT from [KK13] and break it down to 1-out-of-4 bit OT, which again reduces communication at the cost of increased computation (cf. §7.4).

In order to generate a multiplication triple, we first introduce the f^{ab} functionality that is implemented in Protocol 2 using R-OT. The f^{ab} functionality is defined as follows:

$$f^{ab}(\lambda, \lambda) = ((b, v), (a, u)) \in_R \{0,1\}^4 \text{ s.t. } ab = u \oplus v \ .$$

The implementation of this functionality is as follows.

PROTOCOL 2 (Implementing f^{ab} in the R-OT hybrid model)

```
1: $P_S$ and $P_R$ perform a R-OT with $P_S$ as sender and $P_R$ as receiver.
   $P_S$ obtains bits $x_0, x_1$ and $P_R$ obtains random choice bit $a$ and $x_a$ as output.
2: $P_R$ sets $u = x_a$; $P_S$ sets $b = x_0 \oplus x_1$ and $v = x_0$.
   [Note that $ab = u \oplus v$ as $ab = a(x_0 \oplus x_1) = (a(x_0 \oplus x_1) \oplus x_0) \oplus x_0 = x_a \oplus x_0 = u \oplus v$.]
3: $P_R$ outputs $(a, u)$ and $P_S$ outputs $(b, v)$.
```

Note that in Protocol 2, the parties do not send any messages, they just invoke the R-OT functionality and "translate" its output. The security of this protocol is shown via the following argument. There exists, in fact, a bijective function between f^{ab} and the R-OT functionalities, and therefore the security of the two is equivalent both in the presence of a semi-honest adversary. Protocol 2 is in fact a transformation from R-OT to f^{ab}. A transformation from f^{ab} to R-OT can be shown as follows: Given (a, u), the receiver just outputs them both. Given (b, v), the receiver outputs $(v, b \oplus v)$. Since these two functionalities are equivalent, a secure protocol for computing R-OT implies secure protocol for f^{ab}, and a secure protocol for f^{ab} implies secure protocol for R-OT.

We are now ready to implement the f^{mult} functionality in the f^{ab}-hybrid model:

PROTOCOL 3 (Implementing f^{mult} in the f^{ab}-hybrid model)

1: The parties invoke the f^{ab} functionality where P_0 obtains (b_0, v_0) and P_1 obtains (a_1, u_1). Note that $a_1 b_0 = u_1 \oplus v_0$.

2: The parties invoke the f^{ab} functionality where P_0 obtains (a_0, u_0) and P_1 obtains (b_1, v_1). Note that $a_0 b_1 = u_0 \oplus v_1$.

3: Each party outputs $c_i = a_i b_i \oplus u_i \oplus v_i$, and outputs (a_i, b_i, c_i).

Claim 2.2 *Protocol 3 securely computes the f^{mult}-functionality in the f^{ab}-hybrid model, both in presence of a static (probabilistic polynomial time) semi-honest adversary.*

Proof Sketch: Regarding correctness, note that:

$$\begin{aligned} c_0 \oplus c_1 &= (a_0 b_0 \oplus u_0 \oplus v_0) \oplus (a_1 b_1 \oplus u_1 \oplus v_1) = a_0 b_0 \oplus (u_0 \oplus v_1) \oplus (u_1 \oplus v_0) \oplus a_1 b_1 \\ &= a_0 b_0 \oplus a_0 b_1 \oplus a_1 b_0 \oplus a_1 b_1 = (a_0 \oplus a_1)(b_0 \oplus b_1) \ . \end{aligned}$$

Regarding simulation, assume that P_0 is corrupted. The simulator receives as input (a_0, b_0, c_0, v_0) and has to produce the view of the corrupted party, i.e., the messages (b_0, v_0) and (a_0, u_0). It sets $u_0 = c_0 \oplus a_0 b_0 \oplus v_0$, and thus the view is a deterministic function of the output of P_0 (which is the input of the simulator). The simulation is perfect. The case of corrupted P_1 is shown analogously.

■

3 Related Work

In this section, we review related work on semi-honest OT extension (§3.1) and malicious OT extension (§3.2).

3.1 Semi-Honest OT Extension

In the semi-honest model, the protocol of [IKNP03] was implemented by the FastGC framework [HEKM11]. In [HS13], the memory footprint of the OT extension implementation in [HEKM11] was improved by splitting the OT extension protocol sequentially into multiple rounds and speedups were obtained by instantiating the pseudorandom generator with AES instead of SHA-1. In [KK13], a 1-out-of-N OT extension protocol was introduced that is based on the OT extension protocol of [IKNP03] and, for 1-out-of-2 OT on short strings, achieves sub-linear communication in the number of OTs. In particular, for 1-out-of-2 OT on 1-bit strings, their protocol improves communication compared to [IKNP03] by factor 1.6. This improvement in communication comes with an increased cost in computation, since the number of evaluations of the random oracle H for the sender is increased from $2\log_2(N)$ to N. In §7.4 we compare our protocols to [KK13] for 1-out-of-2 OT on 1-bit strings in order to evaluate this computation / communication trade-off. However, we would like to point out that our work is orthogonal to theirs, since our OT protocols maintain their efficiency when obliviously transferring long strings in a 1-out-of-2 OT while their work achieves better efficiency when performing 1-out-of-N OT.

The above works all consider the concrete efficiency of OT extensions. The theoretical feasibility of OT extensions was established in [Bea96], and further theoretical foundations were laid in [LZ13]. [IKOS08] introduced a non-black-box technique for extending OTs with asymptotic constant computation / communication overhead. Their protocol assumes the existence of a polynomial stretch pseudo-random generator in NC^0, i.e., the set of functions that can be computed by a constant depth circuit with bounded fan-in where each output bit depends on a constant number of input bits. The high level idea of the protocol is to use the PRG in the scheme for extending OTs of [Bea96]. However, their scheme is extremely costly in concrete terms and the security of the PRG in NC^0 requires non-standard security assumptions.

3.2 Malicious OT Extension

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 malicious adversaries for OT extensions. All of the known constructions build on the semi-honest protocol of [IKNP03], and add *consistency checks* of different types to the OT extension protocol, to ensure that the receiver sent consistent values. (Note that in [IKNP03], 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 [IKNP03]. This cut-and-choose technique achieves a security of $2^{-\rho}$ by performing ρ parallel evaluations of the basic OT extension protocol.

This was improved on by [Nie07, HIKN08], 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 [IKNP03] for a reasonable number of OTs (with $S = 2$ and approximately 10^7 OTs, they achieve security except with probability 2^{-25}, cf. [Nie07]).

An alternative approach for achieving actively secure OT extension was given in [NNOB12]. Their approach also works in the random oracle model but, instead of performing checks per extended OT as in [Nie07, HIKN08], 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 in the base-OTs with probability 2^{-p}. [NNOB12] shows that this leakage can be tolerated by increasing the number of base-OTs from κ to $\lceil \frac{8}{3}\kappa \rceil$. The [NNOB12] protocol has been optimized and implemented on a GPU in [FN13]. We give a full description of the [NNOB12] protocol with optimizations of [FN13] in Appendix §A.

An approach for achieving actively secure OT extension that works in the standard model has recently been introduced in [Lar14]. Their approach achieves less overhead in the number of base-OTs at the expense of substantially more communication during the check routine (cf. Table 1 on page 5) and is therefore considerably less efficient. Nevertheless, we point out that the work of [Lar14] 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 [NNOB12]. Our approach reduces the number of base-OTs by removing the "sacrifice" step of [NNOB12] (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 [NNOB12]. 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 $60\% - 90\%$ more time and 50% more communication than the passively secure OT extension protocol of [IKNP03] in the LAN- and WAN setting compared to $90\% - 175\%$ more time and 166% more communication for [NNOB12] (cf. Table 1).

In [IPS08] it was shown how to achieve actively secure OT extension with constant overhead from the passively secure protocol of [IKNP03]. Their approach involves the sender and receiver "simulating" additional parties and then running an outer secure computation protocol with security against honest majority. In addition, they show that their transformation can make black-box use of any passively secure OT protocol. Overall, this approach improves on the asymptotic communication of [HIKN08] but the exact constants involved in this approach have not been analyzed.

4 Faster Semi-Honest OT

In the following we describe algorithmic optimizations that improve the scalability and computational complexity of OT extension protocols. We identified computational bottlenecks in OT extension by micro-benchmarking the 1-out-of-2 OT extension implementation of [SZ13].[3] We found that the combined computation time of P_S and P_R was mostly spent on two operations: the matrix transposition (61%) and the evaluation of H, implemented with SHA-256 (32%). (The remaining time was mostly spent on XOR operations (5%) and the evaluation of G, implemented with AES (2%)). Furthermore, for networks with low bandwidth, the communication of OT quickly became the bottleneck. To speed up OT extension, we propose to use parallelization (§4.1), an efficient algorithm for bit-matrix transposition (§4.2), and a protocol optimization that allows to reduce the communication from P_R to P_S by half (§4.3). Note that these implementation optimizations are of general nature and can be applied to our, but also to other OT extension protocols with security against stronger active adversaries.

4.1 Blockwise Parallelized OT Extension

Previous OT extension implementations [CHK$^+$12, SZ13] improved the performance of OT extension by using a *vertical* pipelining approach, i.e., one thread is associated to each step of the protocol: the first thread evaluates the pseudorandom generator G and the second thread evaluates the correlation robust function H (cf. §2.4). However, as evaluation of G is faster than evaluation of H, the workload between the two threads is distributed unequally, causing idle time for the first thread. Additionally, this method for pipelining is designed to run exactly two threads and thus

[3]Note that the implementation in [SZ13] performs 1-out-of-4 OT, but we adapted their implementation since our protocol optimizations target 1-out-of-2 OT extension.

cannot easily be scaled to a larger number of threads.
As observed in [IKNP03, HS13], a large number of OT extensions can be performed by *sequentially* running the OT extension protocol on blocks of fixed size. This reduces the total memory consumption at the expense of more communication rounds.
We propose to use a *horizontal* pipelining approach that splits the matrices processed in the OT extension protocol into *independent* blocks that can be processed in parallel using multiple threads with equal workload, i.e., each of the N threads evaluates the OT extension protocol for $\frac{m}{N}$ inputs in parallel. Each thread uses a separate socket to communicate with its counterpart on the other party, s.t. network scheduling is done by the operating system.

4.2 Efficient Bit-Matrix Transposition

The computational complexity of cryptographic protocols is often measured by counting the number of invocations of cryptographic primitives, since their evaluation often dominates the overall run-time. However, non-cryptographic operations can also have a high impact on the overall run-time of executions although they might seem insignificant in the protocol description. Matrix transposition is an example for such an operation. It is required during the OT extension protocol to transpose the $m \times \ell$ bit-matrix T (cf. §2.4), which is created column-wise but hashed row-wise. Although transposition is a seemingly trivial operation, it has to be performed individually for each entry in T, making it a very costly operation.
We propose to efficiently implement the matrix transposition using Eklundh's algorithm [Ekl72], which uses a divide-and-conquer approach to recursively swap elements of adjacent rows (cf. Figure 1). This decreases the number of swap operations for transposing a $n \times n$ matrix from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log_2 n)$. Additionally, since we process a bit-matrix, we can perform multiple swap operations in parallel by loading multiple bits into one register. Thereby, we again reduce the number of swap operations from $\mathcal{O}(n \log_2 n)$ to $\mathcal{O}(\lceil \frac{n}{r} \rceil \log_2 n)$, where r is the register size of the CPU ($r = 64$ for the machines used in our experiments). Jumping ahead to the evaluation in §7.2, this reduced the total time for the matrix transposition by approximately a factor of 22 from 17.4 s to 0.8 s per party for 2^{24} OTs and reduced the total time for the OTs from 30.4 s to 13.2 s when using a single thread.

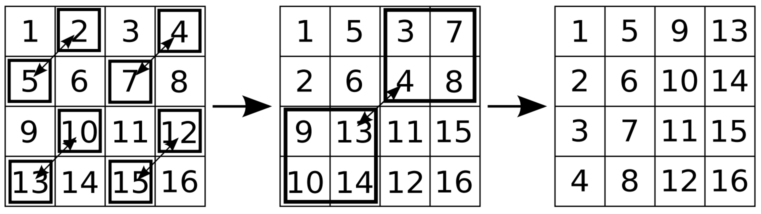


Figure 1: Efficient matrix transposition of a 4×4 matrix using Eklundh's algorithm.

4.3 Optimized Semi-Honest OT Extension

In the following, we optimize the $m \times OT_n$ extension protocol of [IKNP03], described in §2.4. Note that this optimization was independently outlined in [KK13]. Recall, that in the first step of the protocol in [IKNP03], P_R chooses a huge $m \times \ell$ matrix $T = [\mathbf{t}^1 | \dots | \mathbf{t}^\kappa]$ while P_S waits idly (for the semi-honest OT extension protocol we can set $\ell = \kappa$; for the malicious OT extension protocol, ℓ needs to be increased). The parties then engage in a $\ell \times OT_m$ protocol, where the inputs of the receiver are $(\mathbf{t}^i, \mathbf{t}^i \oplus \mathbf{r})$ where $\mathbf{r}$ is its input in the outer $m \times OT_n$ protocol (m selection bits). After the OT, P_S holds $\mathbf{t}^i \oplus (s_i \cdot \mathbf{r})$ for every $1 \leq i \leq \ell$. As described in the appendices of [IKNP03, HMEK11], the protocol can be modified such that P_R only needs to choose two small $\ell \times \kappa$ matrices $K_0 = [\mathbf{k}_1^0 | \ldots | \mathbf{k}_\ell^0]$ and $K_1 = [\mathbf{k}_1^1 | \ldots | \mathbf{k}_\ell^1]$ of seeds. These seeds are used as input to $\ell \times OT_\kappa$; specifically P_R's input as sender in the i-th OT is $(\mathbf{k}_i^0, \mathbf{k}_i^1)$ and, as in [IKNP03], the input of P_S is s_i. To transfer the m-bit tuple $(\mathbf{t}^i, \mathbf{t}^i \oplus \mathbf{r})$ in the i-th OT, P_R expands $\mathbf{k}_i^0$ and $\mathbf{k}_i^1$ using a pseudorandom generator G, sends $(\mathbf{u}^{(i,0)}, \mathbf{u}^{(i,1)}) = (G(\mathbf{k}_i^0) \oplus \mathbf{t}^i, G(\mathbf{k}_i^1) \oplus \mathbf{t}^i \oplus \mathbf{r})$, and P_S recovers $G(\mathbf{k}_i^{s_i}) \oplus \mathbf{u}^{(i,s_i)}$.

Our main observation is that, instead of choosing $\mathbf{t}^i$ randomly, we can set $\mathbf{t}^i = G(\mathbf{k}_i^0)$. Now, P_R needs to send only one m-bit element $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ to P_S (whereas in previous protocols of [IKNP03, HMEK11] two m-bit elements were sent). Observe that if P_S had input $s_i = 0$ in the i-th OT, then it can just define its output $\mathbf{q}^i$ to be $G(\mathbf{k}_i^0) = G(\mathbf{k}_i^{s_i})$. In contrast, if P_S had input $s_i = 1$ in the i-th OT, then it can define its output $\mathbf{q}^i$ to be $G(\mathbf{k}_i^1) \oplus \mathbf{u}^i = G(\mathbf{k}_i^{s_i}) \oplus \mathbf{u}^i$. Since $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, we have that $G(\mathbf{k}_i^1) \oplus \mathbf{u}^i = G(\mathbf{k}_i^0) \oplus \mathbf{r} = \mathbf{t}^i \oplus \mathbf{r}$, as required. The full description of our protocol is given in Protocol 4. This optimization is significant in applications of $m \times OT_n$ extension where m is very large and n is short, such as in GMW. In typical use-cases for GMW, m is in the size of several millions to a billion (cf. examples in §1), while n is one. Thereby, the communication complexity of GMW is almost reduced by half.

In addition, observe that the initial OT phase in Protocol 4 is completely independent of the actual inputs of the parties. Thus, the parties can compute the initial base-OTs before their inputs are determined.

Finally, another problem that arises in the original protocol of [IKNP03] is that the entire $m \times \ell$ matrix is transmitted together and processed. This means that the number of OTs to be obtained must be predetermined and, if m is very large, this results in considerable latency as well as memory management issues. As in [HS13], splitting the matrix into smaller blocks that are processed in a pipelined fashion reduces latency, computation time, and avoids memory management problems. In addition, it is possible to continually extend OTs, with no a priori bound on m. This is very useful in a secure computation setting, where parties may interact many times together with no a priori bound. We state and prove security of our optimizations next.

Theorem 4.1 *Assuming that G is a pseudorandom generator and H is a correlation-robust function (as in Definition 2.1), Protocol 4 correctly and privately computes the $m \times OT_n$-functionality in the presence of semi-honest adversaries, in the $\ell \times OT_\kappa$-hybrid model.*

Proof: We first show that the protocol implements the $m \times OT_n$-functionality. Then, we prove that the protocol is secure where the sender is corrupted, and finally that it is secure when the receiver is corrupted.

Correctness. We show that the output of the receiver is $(x_1^{r_1}, \ldots, x_m^{r_m})$ in an execution of the protocol where the inputs of the sender are $((x_1^0, x_1^1), \ldots, (x_m^0, x_m^1))$ and the input of the receiver is $\mathbf{r} = (r_1, \ldots, r_m)$. We have two cases:

1. $\boldsymbol{r_j = 0}$**:** Recall that $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$, and so $\mathbf{q}_j = \mathbf{t}_j$. Thus:

$$\begin{aligned} x_j &= y_j^0 \oplus H(j, \mathbf{t}_j) = x_j^0 \oplus H(j, \mathbf{q}_j) \oplus H(j, \mathbf{t}_j) \\ &= x_j^0 \oplus H(j, \mathbf{t}_j) \oplus H(j, \mathbf{t}_j) = x_j^0 \end{aligned}$$

PROTOCOL 4 (Optimized semi-honest secure OT extension protocol)

- **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 $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Common Input:** Symmetric security parameter κ and number of base-OTs $\ell = \kappa$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the $\ell \times OT_\kappa$ functionality and use a 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 Def. 2.1).

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 $\ell \times OT_\kappa$-oracle, 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 $T = [\mathbf{t}^1 | \ldots | \mathbf{t}^\ell]$ 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]*:*
   (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 $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. (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 $1 \le j \le m$, 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 $1 \le j \le m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j)$.
3. **Output:** P_R outputs $(x_1, \ldots, x_m)$; P_S has no output.

[a]This phase can be iterated. Specifically, R can compute the next κ bits of $\mathbf{t}^i$ and $\mathbf{u}^i$ (by applying G to get the next κ bits from the PRG for each of the seeds and using the next κ bits of its input in $\mathbf{r}$) and send the block of $\kappa \times \kappa$ bits to S (κ bits from each of $\mathbf{u}^1, \ldots, \mathbf{u}^\kappa$).

2. $\boldsymbol{r_j = 1}$**:** In this case $\mathbf{q}_j = \mathbf{s} \oplus \mathbf{t}_j$, and so:

$$\begin{aligned} x_j &= y_j^1 \oplus H(j, \mathbf{t}_j) = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s}) \oplus H(j, \mathbf{t}_j) \\ &= x_j^1 \oplus H(j, \mathbf{t}_j) \oplus H(j, \mathbf{t}_j) = x_j^1 \end{aligned}$$

Corrupted Sender. The view of the sender during the protocol contains the output from the $\ell \times OT_\kappa$ invocation and the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$. The simulator $\mathcal{S}_0$ simply outputs a uniform string $\mathbf{s} \in \{0,1\}^\ell$ (which is the only randomness that P_S chooses in the protocol, and therefore w.l.o.g. can be interpreted as the random tape of the adversary), ℓ random seeds $\mathbf{k}_1^{s_1}, \ldots, \mathbf{k}_\ell^{s_\ell}$, which are chosen uniformly from $\{0,1\}^\kappa$, and ℓ random strings $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$, chosen uniformly from $\{0,1\}^m$. In the real execution, $(\mathbf{s}, \mathbf{k}_1^{s_1}, \ldots, \mathbf{k}_\ell^{s_\ell})$ are chosen in exactly the same way. Each value $\mathbf{u}^i$ for $1 \le i \le \ell$ is defined as $G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$. Since $\mathbf{k}_i^{1-s_i}$ is unknown to P_S (by the security of the $\ell \times OT_\kappa$ functionality), we have that $G(\mathbf{k}_i^{1-s_i})$ is indistinguishable from uniform, and so each $\mathbf{u}^i$ is indistinguishable from uniform. Therefore, the view of the corrupted sender in the simulation is indistinguishable from its view in a real execution.

Corrupted Receiver. The view of the corrupted receiver consists of its random tape and the messages $((y_1^0, y_1^1), \ldots, (y_m^0, y_m^1))$ only. The simulator $\mathcal{S}_1$ is invoked with the inputs and outputs of the receiver, i.e., $\mathbf{r} = (r_1, \ldots, r_m)$ and $(x_1^{r_1}, \ldots, x_m^{r_m})$. $\mathcal{S}_1$ then chooses a random tape ρ for the adversary (which determines the $\mathbf{k}_i^0, \mathbf{k}_i^1$ values), defines the matrix T, and computes $y_j^{r_j} = x_j^{r_j} \oplus H(j, \mathbf{t}_j)$ for $1 \leq j \leq m$. Then, it chooses each $y_j^{1-r_j}$ uniformly and independently at random from $\{0,1\}^n$. Finally, it outputs $(\rho, (y_1^0, y_1^1), \ldots, (y_m^0, y_m^1))$ as the view of the corrupted receiver.
We now show that the output of the simulator is indistinguishable from the view of the receiver in a real execution. If $r_j = 0$, then $\mathbf{q}_j = \mathbf{t}_j$ and thus $(y_j^0, y_j^1) = (x_j^0 \oplus H(j, \mathbf{t}_j), x_j^1 \oplus H(j, \mathbf{t}_j \oplus \mathbf{s}))$. If $r_j = 1$, $\mathbf{q}_j = \mathbf{t}_j \oplus \mathbf{s}$ and therefore $(y_j^0, y_j^1) = (x_j^0 \oplus H(j, \mathbf{t}_j \oplus \mathbf{s}), x_j^1 \oplus H(j, \mathbf{t}_j))$. In the simulation, the values $y_j^{r_j}$ are computed as $x_j^{r_j} \oplus H(j, \mathbf{t}_j)$ and therefore are identical to the real execution. It therefore remains to show that the values $(y_1^{1-r_1}, \ldots, y_m^{1-r_m})$ as computed in the real execution are indistinguishable from random strings as output in the simulation. As we have seen, in the real execution each $y_j^{1-r_j}$ equals $x_j^{1-r_j} \oplus H(j, \mathbf{t}_j \oplus \mathbf{s})$. Since H is a correlation robust function, it holds that:

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

for random $\mathbf{s}, \mathbf{t}_1, \ldots, \mathbf{t}_m \in \{0,1\}^\ell$, where U_a defines the uniform distribution over $\{0,1\}^a$ (see Definition 2.1). In the protocol we derive the values $\mathbf{t}_1, \ldots, \mathbf{t}_m$ by applying a pseudorandom generator G to the seeds $\mathbf{k}_1^0, \ldots, \mathbf{k}_\ell^0$ and transposing the resulting matrix. We need to show that the values $H(j, \mathbf{t}_1 \oplus \mathbf{s}), \ldots, H(j, \mathbf{t}_m \oplus \mathbf{s})$ are still indistinguishable from uniform in this case. However, this follows from a straightforward hybrid argument (namely, that replacing truly random $\mathbf{t}^i$ values in the input to H with pseudorandom values preserves the correlation robustness of H). We conclude that the ideal and real distributions are computationally indistinguishable. ■

5 Faster Malicious OT

On the Malicious Security of [IKNP03]. 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 [IKNP03] 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. [IKNP03]). 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 $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$.

5.1 Overview of our Malicious Secure 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 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 $\mathbf{k}_i^{\overline{s_i}}, \mathbf{k}_j^{\overline{s_j}}$). P_R cannot give these values to P_S since this will reveal its choice bits. However, P_R can send the hashes of these inverse values. Specifically, the P_R commits 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 set $h_{i,j}^{\overline{s_i},\overline{s_j}} = H(G(\mathbf{k}_i^{\overline{s_i}}) \oplus G(\mathbf{k}_j^{\overline{s_j}}) \oplus \mathbf{r}^i \oplus \mathbf{r}^j)$, 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 4. We give some more explanations regarding the possibility of the adversary to cheat during the consistency check in §5.2.

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 $\{H(j, \mathbf{t}_j \oplus \mathbf{s})\}_j$ 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 $H(j, \mathbf{t}_j \oplus \mathbf{s})$ 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 [NNOB12] 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 (§5.2). We then show how to reduce the number of consistency checks to achieve better performance (§5.3), and how to replace the random oracle with a weaker correlation robustness assumption (§5.4). Finally, we show how our protocol can be used to achieve covert security (§5.5).

PROTOCOL 5 (Our actively 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 the number of base-OTs is $\ell = \kappa + \rho$.
- **Oracles and cryptographic primitives:** The parties use an ideal $\ell \times OT_\kappa$ functionality, pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^{m+\kappa}$, and random-oracle H (see §5.4 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 $\ell \times OT_\kappa$-oracle, 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 $(m+\kappa) \times \ell$ 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 chooses a random string $\boldsymbol{\tau} \in \{0,1\}^\kappa$, and defines $\mathbf{r}' = \mathbf{r} || \boldsymbol{\tau}$.
   (b) 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 $\perp$.
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 $(m+\kappa) \times \ell$ 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 $(x_1, \ldots, x_m)$; P_S has no output.

5.2 The Security of Our Protocol

Malicious sender. The original OT extension protocol of [IKNP03] already provides security against a malicious P_S. In an earlier version of this paper, we claimed that our checks in Step 3 of the protocol do not provide a malicious P_S with any new capabilities, and we derived security directly from the semi-honest case with no further proof.

However, as pointed out by [Sch17], adding the checks for consistency of $\mathbf{r}$ provides an "oracle" for checking whether a particular candidate $\tilde{\mathbf{r}}$ is the input of the receiver or not. In fact, given $\mathbf{u}^\alpha$, $G(\mathbf{k}_\alpha^{s_\alpha})$ and $G(\mathbf{k}_\beta^{s_\beta})$ (note that $\mathbf{k}_\alpha^{s_\alpha}$ and $\mathbf{k}_\beta^{s_\beta}$ are sent to the sender during the base OT stage), the malicious sender can check whether $\mathbf{r} = \tilde{\mathbf{r}}$ or not, by first computing $\mathbf{t}_\alpha^{\overline{s_\alpha}} = \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^{s_\alpha}) \oplus \tilde{\mathbf{r}}$ and then checking whether $h_{\alpha,\beta}^{\overline{s_\alpha},s_\beta} = H(\mathbf{t}_\alpha^{\overline{s_\alpha}} \oplus G(\mathbf{k}_\beta^{s_\beta}))$.

In order to prevent this subtle attack, we modify the protocol such that P_R first appends to $\mathbf{r}$ some random string $\boldsymbol{\tau} \in \{0,1\}^\kappa$ resulting in a string $\mathbf{r}' = \mathbf{r}||\boldsymbol{\tau}$ (this is Step 2a in the protocol). This adds entropy to the choice of $\mathbf{r}$ and therefore prevents the above attack. Note that in the second part of the OT extension phase, the transfer is done on only on the first m rows of the matrix Q and not $m + \kappa$. In Appendix B we formally prove that the protocol is secure in the presence of a malicious sender, assuming that the function H is modeled as κ-min-entropy strongly correlation robust (see §5.11 regarding this assumption). We also note that the change in the efficiency of the protocol is minor, as $m \gg \kappa$.

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 [IKNP03]. 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 $1 \le i \le \ell$, 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.

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 $|\mathcal{U}| \ge \ell - \rho$. Thus, for every $1 \le j \le m$, 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 $(\mathbf{u}^1, \dots, \mathbf{u}^\ell)$ 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 $\mathcal{T} = (\mathcal{K}, \mathbf{u}^1, \dots, \mathbf{u}^\ell, \mathcal{H})$ 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.

In the following, we let $\mathcal{T}_{\alpha,\beta}$ denote all messages that the receiver sends and which are relevant for to the verification of the pair (α, β), that is, $\mathcal{T}_{\alpha,\beta} = \left(\mathbf{k}_\alpha^0, \mathbf{k}_\alpha^1, \mathbf{k}_\beta^0, \mathbf{k}_\beta^1, \mathbf{u}^\alpha, \mathbf{u}^\beta, \mathcal{H}_{\alpha,\beta}\right)$. Note that $\mathcal{T}$, the set of all messages that the receiver sends, is defined as $\mathcal{T} = \bigcup_{\alpha,\beta} \mathcal{T}_{\alpha,\beta} = (\mathcal{K}, \mathbf{u}^1, \dots, \mathbf{u}^\ell, \mathcal{H})$, exactly as considered above.

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 5.1 *Let $\mathcal{T}_{\alpha,\beta} = \{\{\mathbf{k}_\alpha^b\}_b, \{\mathbf{k}_\beta^b\}_b, \mathbf{u}^\alpha, \mathbf{u}^\beta, \mathcal{H}_{\alpha,\beta}\}$ and assume that H is a collision-resistant hash-function. Then, the following holds, except with negligible probability::*

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, except for some negligible probability 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 $\mathbf{r}^\alpha \overset{\text{def}}{=} \mathbf{u}^\alpha \oplus G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\alpha^1)$, and $\mathbf{r}^\beta \overset{\text{def}}{=} \mathbf{u}^\beta \oplus G(\mathbf{k}_\beta^0) \oplus G(\mathbf{k}_\beta^1)$. 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 5.1 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: $\mathbf{r}^{\boldsymbol{\alpha}} = \mathbf{r}^{\boldsymbol{\beta}}$ and $\boldsymbol{\mathcal{H}_{\alpha,\beta}}$ is correct.** That is, for every $(a,b) \in \{0,1\}^2$: $h_{\alpha,\beta}^{a,b} = H\left(G(\mathbf{k}_\alpha^a) \oplus G(\mathbf{k}_\beta^b)\right)$. In this case, 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 5.1, 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 $h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}$ correctly (i.e., $h_{\alpha,\beta}^{0,0} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^0))$ and $h_{\alpha,\beta}^{0,1} = H(G(\mathbf{k}_\alpha^0) \oplus G(\mathbf{k}_\beta^1)))$ and set values $h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1}$, accordingly, such that the verification will pass for the cases of $(s_\alpha, s_\beta) = (0,0)$ or $(0,1)$. That is, it sets:

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

   (and it sets $h_{\alpha,\beta}^{1,1}$ 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.

   In case where $\mathcal{H}_{\alpha,\beta}$ is correct in only one position but $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$, the probability of success is even smaller. For instance, assume that $h_{\alpha,\beta}^{a,b} = H(G(\mathbf{k}_\alpha^a) \oplus G(\mathbf{k}_\beta^b))$ for $(a,b) = (0,0), (0,1), (1,0)$, but the adversary sends $h_{\alpha,\beta}^{1,1}$ incorrectly as above. In this case, the verification will fail for $(s_\alpha, s_\beta) = (1,1)$ and, in addition, also for the cases where $(s_\alpha, s_\beta) = (0,1)$ or $(1,0)$, since $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$. Similarly, for the case where $\mathcal{H}_{\alpha,\beta}$ is incorrect in only one position, for which the adversary only succeeds with probability $1/2$. Therefore, it is more beneficial for the adversary to send two positions incorrectly.

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 $h_{\alpha,\beta}^{a,b}$ correctly, set $h_{\alpha,\beta}^{\overline{a},\overline{b}}$ 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 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 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 $1/2$.

Proving security of the protocol. Before proceeding to the full proof of security, we first provide a 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 $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$. 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 5, 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 $\{y_j^{\overline{r_j}}\}_{j=1}^m$.

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, $|S| \geq 2^\kappa$). We now proceed to a formal statement of the theorem and formal proof of security, where there we prove the two informal claims that were just mentioned.

Theorem 5.2 *Assume that H is a random oracle and that G is a pseudorandom generator. Then, Protocol 5 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: Recall that security against malicious sender can be proven by a simple reduction to the original OT extension protocol of [IKNP03], which is already secure against malicious sender, using the fact that our checks consist of messages that go from the receiver to the sender only. In the following we will give the proof for a malicious receiver. Since we already gave some proof sketch, we start directly with a formal description of the simulator $\mathcal{S}$:

The simulator $\mathcal{S}$.

1. The simulator invokes the adversary $\mathcal{A}$ on the auxiliary input z.
2. **Initial OT phase:** The adversary $\mathcal{A}$ outputs ℓ pairs of κ-bits each $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$ as input to the $\ell \times OT_\kappa$-functionality. It receives no output from this invocation.
3. **First part of OT extension phase:** The adversary $\mathcal{A}$ outputs ℓ strings $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$.
4. **Consistency check of r:**
   (a) For every $\alpha, \beta \in [\ell]^2$, the adversary $\mathcal{A}$ outputs the quadruple $H^{\alpha,\beta} = (h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}, h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1})$.
   (b) The simulator chooses a string $\mathbf{s}$ uniformly at random from $\{0,1\}^\ell$.
   (c) Given the values $\{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \{H^{\alpha,\beta}\}_{\alpha,\beta}\}$ and the chosen secret $\mathbf{s}$, the simulator can perform all the needed checks as the honest sender in the real execution. In case where one of the verification fails, the simulator halts.
5. **Second part of the OT extension phase:**
   (a) The simulator computes the matrices T, Q and R, where for every i, $\mathbf{t}^i = G(\mathbf{k}_i^0)$, $\mathbf{q}^i = (s_i \cdot \mathbf{r}^i) \oplus \mathbf{t}^i$ and $\mathbf{r}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{u}^i$.
   (b) From all the vectors $\mathbf{r}^1, \ldots, \mathbf{r}^\ell$, let $\mathbf{r}$ be the vector that is mostly repeated (as we will see, the verification process guarantees that there exists a vector that is repeated at least $\ell - \rho$ times).
   (c) Send $\mathbf{r}$ to the trusted party, and receive $x_1^{r_1}, \ldots, x_m^{r_m}$. Define the values $\mathbf{e}_j$ for every $1 \leq j \leq m$ (explicitly, define the matrix R as the matrix for which its ith column is $\mathbf{r}^i$, and let $\mathbf{r}_j$ denote its jth row. Then, $\mathbf{e}_j = (r_j \cdot \mathbf{1}) \oplus \mathbf{r}_j$. Then, for every $1 \leq j \leq m$, set $y_j^{r_j} = x_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$, and set $y_j^{\overline{r_j}}$ uniformly at random. Send $\{(y_j^0, y_j^1)\}_{j=1}^m$ to the adversary $\mathcal{A}$, output whatever it outputs and halt.

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 OT primitive, we can assume that). We divide all possible choices of $\mathcal{T}$ into two sets, $\mathcal{T}_{\mathsf{good}}$ and $\mathcal{T}_{\mathsf{bad}}$, defined as follows:

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

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 $|\mathcal{S}(\mathcal{T})| > 2^\ell \cdot 2^{-\rho} = 2^{\ell-\rho} = 2^\kappa$. That is, in case a transcript $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ passes the consistency check of $\mathbf{r}$, there are at least 2^κ different secrets $\mathbf{s}$ that are consistent with the given transcript, each are likely with the same probability, and thus the adversary may guess $\mathbf{s}$ with probability at most $2^{-\kappa}$.

Let $\mathcal{U}$ be the largest set of indices such that for every $i, j \in \mathcal{U}$, $\mathbf{r}^i = \mathbf{r}^j$. Let B be the complementary set, that is, $B = [\ell] \setminus \mathcal{U}$. From the definition of the sets, for every $\alpha \in \mathcal{U}$ and $\beta \in B$, it holds that $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$.

We claim that if $|\mathcal{U}| < \ell - \rho$ (i.e., $|B| > \rho$), then it must hold that $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$ and the adversary gets caught with high probability. That is:

Claim 5.3 *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 $|\mathcal{U}| < \ell - \rho$. Then:*

$$\Pr_{\mathbf{s}}[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1] \leq 2^{-\rho}$$

and thus, $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$.

We will prove the claim below. Let $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, and let $\mathcal{U}$ and B be as above. Using the claim above, we have that $|B| < \rho$. We now focus on the set of secrets $\mathbf{s}$ that are consistent with this transcript $\mathcal{T}$, i.e., the set $S(\mathcal{T})$. For a set of indices A, we let $\mathbf{s}_A$ denote all the bits in $\mathbf{s}$ with indices from A, that is $\mathbf{s}_A = (\mathbf{s}_a)_{a \in A}$. We now claim that the bits $\mathbf{s}_B$ are common to all the secrets in $S(\mathcal{T})$, and thus, even when we give the adversary the bits $\mathbf{s}_B$ in the clear once the verification is completed, the adversary still has to guess $\mathbf{s}$ from a set of at least 2^κ secrets. Formally:

Claim 5.4 *Let $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, and let $\mathcal{U}$ and B as above. Then, there exists a string $w \in \{0,1\}^{|B|}$, such that for every $\mathbf{s}' \in S(\mathcal{T})$, it holds that: $\mathbf{s}'_B = w$.*

Proof: From the definition of the sets B and $\mathcal{U}$, it holds that for every $\alpha \in \mathcal{U}$ and $\beta \in B$, $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$. Consider two secrets $\mathbf{s}, \mathbf{s}'$ that are consistent with $\mathcal{T}$ (since $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, there are many such $\mathcal{T}$'s). We show the following:

- If there exists an index $\beta \in B$ such that $s_\beta \neq s'_\beta$, then for every $\alpha \in \mathcal{U}$ it holds that $\mathbf{s}_\alpha = s'_\alpha$ (that is, $\mathbf{s}_{\mathcal{U}} = \mathbf{s}'_{\mathcal{U}}$).
- Similarly, if there exists an index $\alpha \in \mathcal{U}$ such that $s_\alpha \neq s'_\alpha$ then for every $\beta \in B$ it holds that $s_\beta = s'_\beta$ (that is, $\mathbf{s}_B = \mathcal{S}'_B$).

We show the first claim. Assume that $\mathbf{s}_B \neq \mathbf{s}'_B$, thus, there must exist an index $\beta \in B$ such that $s_\beta \neq s'_\beta$, i.e., $s'_\beta = \overline{s_\beta}$. Now, consider some $\alpha \in \mathcal{U}$, we show that $s_\alpha = s'_\alpha$ and thus $\mathbf{s}_{\mathcal{U}} = \mathbf{s}'_{\mathcal{U}}$. Recall that $\mathcal{T}$ is consistent with $\mathbf{s}$, and therefore is consistent with (s_α, s_β). From Lemma 5.1, it is *inconsistent* with $(\overline{s_\alpha}, \overline{s_\beta}) = (\overline{s_\alpha}, s'_\beta)$. However, recall that $\mathcal{T}$ is consistent also with $\mathbf{s}'$, and therefore it is consistent with (s'_α, s'_β). We therefore conclude that it must hold that $\overline{s_\alpha} \neq s'_\alpha$ and thus $s_\alpha = s'_\alpha$. The second claim is proven analogously.

We now claim that the set $S(\mathcal{T})$ either shares the same $\mathbf{s}_B$ for all its elements, or shares the same $\mathbf{s}_{\mathcal{U}}$ for all elements (and of course, not both). In order to see this, let $S(\mathcal{T}) = \{\mathbf{s}^1, \ldots, \mathbf{s}^n\}$. Assume, without loss of generality, that $\mathbf{s}^1_{\mathcal{U}} \neq \mathbf{s}^2_{\mathcal{U}}$ (and so, from above, $\mathbf{s}^1_B = \mathbf{s}^2_B$). We now claim that all the other elements share the same bits in B. If not, that is, if there exists an element $\mathbf{s}^i \in \mathcal{S}(\mathcal{T})$ such that $\mathbf{s}^i_B \neq \mathbf{s}^1_B$, it must hold that $\mathbf{s}^i_{\mathcal{U}} = \mathbf{s}^1_{\mathcal{U}}$. However, $\mathbf{s}^1_{\mathcal{U}} \neq \mathbf{s}^2_{\mathcal{U}}$, which implies that $\mathbf{s}^i_{\mathcal{U}} \neq \mathbf{s}^2_{\mathcal{U}}$ and thus $\mathbf{s}^i_B = \mathbf{s}^2_B = \mathbf{s}^1_B$, in contradiction.

We further claim that the set $S(\mathcal{T})$ must share the same $\mathbf{s}_B$ and cannot share the same $\mathbf{s}_{\mathcal{U}}$. This is a simple counting argument: Since $|B| < \rho$, a set $S(\mathcal{T})$ that shares the same $\mathbf{s}_{\mathcal{U}}$ has size of at most $2^{|B|} \leq 2^{\rho}$. However, since $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, it holds that $|S(\mathcal{T})| > 2^{\kappa}$. Therefore, the set must share the same $\mathbf{s}_B$, and the claim follows. ■

We now show that the distinguisher distinguishes between the ideal and real executions with relatively small probability, even when it asks the oracle H as (polynomially) many queries as it wishes.

First, assume that the distinguisher cannot make any queries to H. We claim that the distributions of the real and ideal executions are statistically close. Intuitively, if the adversary outputs $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$, then clearly the distinguisher may succeed only if the consistency check fails, which happens with probability at most $2^{-\rho}$. On the other hand, in case where the adversary outputs $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, then, except for negligible probability (in ρ) it holds that $|\mathcal{U}| \geq \ell - \rho = \kappa$, and $|\mathcal{S}(\mathcal{T})| \geq 2^{\kappa}$, where all the elements in $S(\mathcal{T})$ share the same bits $\mathbf{s}_B$. Thus, even if the distinguisher receives $\mathbf{s}_B$ in the clear, the values $H(j, \mathbf{q}_j)$, $H(j, \mathbf{q}_j \oplus \mathbf{s})$ that are used for masking the inputs are uniformly random and independent of each other. Therefore, the simulation is indistinguishable from the real-execution.

Now, assume that the distinguisher can also make queries to the random oracle H. In this case, we claim that the distinguisher can distinguish only if it makes a "critical query", where:

Definition 5.5 *For every $1 \leq j \leq m$, a query made by the distinguisher or the receiver to the random oracle is a* critical query *if it is of the form $(j, ((1 - r_j) \cdot \mathbf{s}) \oplus \mathbf{q}_j)$ for some j.*

Note that a critical query can also be represented as $H(j, (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j))$. Clearly, a critical query totally reveals $x_j^{\overline{r_j}}$. Conditioned on the event that the distinguisher (or the receiver) never queries such a critical query and that $\mathbf{s}_{\mathcal{U}} \neq 0$, the distributions of the real and ideal executions are statistically close. On the other hand, as long as $\mathbf{s}_{\mathcal{U}} \neq 0$, and as long as no such a critical query is made, the answers to the queries are independent of the value of $\mathbf{s}$, and the distinguisher does not learn anything new from the queries themselves. Any query to H is distributed uniformly and independently at random, and since $\mathbf{s}$ is distributed uniformly in $\mathcal{S}(\mathcal{T})$, the probability to "hit" a critical query is bounded by $1/|\mathcal{S}(\mathcal{T})|$. We therefore conclude the following claim:

Claim 5.6 *The probability that the distinguisher or the receiver make a* critical query *is bounded by: $(t+1)/|\mathcal{S}(\mathcal{T})| \leq (t+1) \cdot 2^{-\kappa}$, where t is an upper-bound on the number of their queries.*

This completes the proof. ■

We now restate Claim 5.3 and prove it. This claim is in fact, an analysis of the consistency check phase of the protocol.

Claim 5.7 (Claim 5.3, restated) *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 $|\mathcal{U}| < \ell - \rho$. Then:*

$$\Pr_{\mathbf{s}}[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1] \leq 2^{-\rho}$$

and thus, $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$.

Proof: Let $\mathcal{T}$ be the values that the adversary outputs, i.e., the values

$$\mathcal{T} = \left\{ \{\mathbf{k}_i^0, \mathbf{k}_i^1\}_i, \{\mathbf{u}^i\}_i, \{\mathcal{H}_{\alpha,\beta}\}_{\alpha,\beta} \right\}$$

For a pair $\alpha \in \mathcal{U}$, $\beta \in B$, we claim that the adversary passes the verification of the pair (α, β) with probability at most $1/2$. This is because $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$ and due to Lemma 5.1, if $\mathcal{T}$ is consistent with some (s_α, s_β) then it is inconsistent with $(\overline{s_\alpha}, \overline{s_\beta})$. Thus, there are at most 2 possible values (s_α, s_β) that are consistent with $\mathcal{T}$, and the adversary gets caught for the 2 other values.

We define the inconsistency graph $\Gamma = (V, E)$ as follows. The set of vertices is the set $[\ell]$. The set of edges contains all the pairs that define different $\mathbf{r}$'s, that is, there exists an edge (α, β) if $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$. Note that since (α, β) are not consistent, the adversary gets caught in the check (α, β) with probability at least $1/2$. We sometimes consider the complement graph (or, the consistency graph) $\overline{\Gamma} = (V, \overline{E})$. In $\overline{\Gamma}$, each edge represents that the two vertices define the same implicit input $\mathbf{r}$.

We now analyze the size of the set $\mathcal{U}$.

1. **Case 1: $\rho \leq |\mathcal{U}| < \ell - \rho$.** In this case, we have a large enough set which is consistent within itself, but is inconsistent with all the others. We claim that in this case, the adversary will get caught with probability $1 - 2^{-\rho}$.

   In order to see this, consider the set $B = [\ell] \setminus \mathcal{U}$. Since $B \cup \mathcal{U} = [\ell]$, we have that $\rho < |B| \leq \ell - \rho$ as well.

   Moreover, consider the inconsistency graph Γ, and remove all the edges that are between two elements of B (this can be interpreted as follows: although there is some possibility that the adversary gets caught because of these checks, we ignore them and do not consider these inconsistencies as cheating). As a result, we have a bipartite graph where the set of vertices is divided to B and $\mathcal{U}$. Moreover, when considering the complement graph for the resulting graph, we have two cliques, B and $\mathcal{U}$, and the maximal clique in this graph is at most $\ell - \rho$.

   According to König's theorem [LP86], in any bipartite graph the number of edges in the maximal matching equals the minimal vertex cover. Moreover, it is easy to see that the sum of the minimum vertex cover in some graph, and the maximal clique of its complement graph equals to the overall number of vertices ℓ. We therefore conclude that the maximal matching in the graph Γ is at least ρ.

   Each edge in the graph represents a check where the adversary gets caught with probability at least $1/2$. Since there are at least ρ edges in the maximal matching in the inconsistency graph, there are at least ρ pairs for which the adversary gets caught with probability at least $1/2$. Moreover, since we have a matching, each pair and check are independent. We therefore conclude that the adversary succeeds in its cheating with probability at most $2^{-\rho}$, and therefore it gets caught with probability at least $1 - 2^{-\rho}$.

2. **Case 2: $|\mathcal{U}| < \rho$.** Similarly to the previous case, we can just find a superset $\mathcal{U}'$ that contains $\mathcal{U}$ of size at least ρ for which we assume (artificially) that is all consistent. That is, for this set $\mathcal{U}'$ we just ignore the checks between the elements in this set and assume that they are all consistent. After we obtain this clique (by ignoring some of the checks), we are back to the previous case.

For conclusion, we have the following: if $\mathcal{T}$ is such that $|\mathcal{U}| < \ell - \rho$, then :

$$\Pr_{\mathbf{s}}[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1] < 2^{-\rho}$$

■

5.3 Reducing the Number of Checks

In Protocol 5, 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 5 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 5 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 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 6 we modify the consistency check of $\mathbf{r}$ that appears in Step 3 of Protocol 5. 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 6 (Modification for Protocol 5, Fewer Checks)
The parties run Protocol 5 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:
$$\begin{aligned} h^{0,0}_{\alpha,\beta} &= H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^0_\beta)) & h^{0,1}_{\alpha,\beta} &= H(G(\mathbf{k}^0_\alpha) \oplus G(\mathbf{k}^1_\beta)) \ , \\ h^{1,0}_{\alpha,\beta} &= H(G(\mathbf{k}^1_\alpha) \oplus G(\mathbf{k}^0_\beta)) & h^{1,1}_{\alpha,\beta} &= H(G(\mathbf{k}^1_\alpha) \oplus G(\mathbf{k}^1_\beta)) \ . \end{aligned}$$
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 5.2, 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 responses that maximizes the probability to pass the verification, that is:

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

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

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

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 at least $1-2^{-\rho}$.

The following claim is the analogue of Claim 5.3, and it 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 5.8 *Let $\mathcal{T}$ be as above, and let $\mathcal{U}$ be the largest set of indices such that for every $\alpha,\beta\in\mathcal{U}$, $\mathbf{r}^{\alpha}=\mathbf{r}^{\beta}$. For appropriate choice of parameters μ,ℓ, for every $\mathcal{T}$ such that $|\mathcal{U}|\leq\kappa$ it holds that:*

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

Proof: The partial transcript $\mathcal{T}$ defines the two sets B and $\mathcal{U}$. Viewing the ℓ base-OTs 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 $(B\cup\mathcal{U},E')$ 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 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 [LP86]). 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 5.9 *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 to address the edges that connect the vertexes in the sets). 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 independence everywhere, we have:

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

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:

$$\begin{aligned}
\Pr\left[\sum_{S,T,\text{ s.t. }(\star)} X_{S,T} > 0\right] &\leq \sum_{S,T\text{ s.t. }(\star)} \Pr\left[X_{S,T} = 1\right] \qquad (2)\\
&\leq \sum_{S,T\text{ s.t. }(\star)} \left(\frac{|B|+|T|}{\ell}\right)^{|S|\cdot\mu} \cdot \left(\frac{\ell - |S|}{\ell}\right)^{(|\mathcal{U}|-|T|)\cdot\mu}\\
&= \sum_{|S|=|B|-\rho}^{|B|} \sum_{|T|=0}^{|S|-|B|+\rho} \binom{|B|}{|S|} \cdot \binom{|\mathcal{U}|}{|T|} \cdot \left(\frac{|B|+|T|}{\ell}\right)^{|S|\cdot\mu} \cdot \left(\frac{\ell-|S|}{\ell}\right)^{(|\mathcal{U}|-|T|)\cdot\mu}
\end{aligned}$$

We proceed to show an asymptotic analysis for the above, and show that it is bounded by $2^{-\rho}$ for appropriate choice of parameters and large enough μ. We remark that we did not attempt to provide a tight asymptotic analysis since for concrete use, we compute the parameters from the exact bound given above; see Table 3.

In the asymptotic analysis, we omit the last term. Since $|B| - \rho \leq |S| \leq |B|$ and $0 \leq |T| \leq |S| - |B| + \rho \leq \rho$ and have that:

$$\left(\frac{|B|+|T|}{\ell}\right)^{|S|\cdot\mu} \leq \left(\frac{|S|+\rho}{\ell}\right)^{|S|\cdot\mu} \leq \left(\frac{|B|+\rho}{\ell}\right)^{(|B|-\rho)\cdot\mu} .$$

Moreover, $\binom{|B|}{|S|} = \frac{|B|!}{|S|!\cdot(|B|-|S|)!}$, where, for every $|B| - \rho \leq |S| \leq |B|$, it holds that $\binom{|B|}{|S|} \leq |B|^{\rho}$. Likewise, $\binom{|\mathcal{U}|}{|T|} \leq |\mathcal{U}|^{\rho}$ for every $0 \leq |T| \leq |S| - |B| + \rho \leq \rho$.

$$\begin{aligned}
\Pr\left[\sum_{S,T,\text{ s.t. }(\star)} X_{S,T} > 0\right] &\leq \sum_{|S|=|B|-\rho}^{|B|} \sum_{|T|=0}^{|S|-|B|+\rho} \binom{|B|}{|S|} \cdot \binom{|\mathcal{U}|}{|T|} \cdot \left(\frac{|B|+|T|}{\ell}\right)^{|S|\cdot\mu} \cdot \left(\frac{\ell-|S|}{\ell}\right)^{(|\mathcal{U}|-|T|)\cdot\mu}\\
&\leq \rho^2 \cdot |B|^{\rho} \cdot |\mathcal{U}|^{\rho} \cdot \left(\frac{|B|+\rho}{\ell}\right)^{(|B|-\rho)\cdot\mu} .
\end{aligned}$$

which is bounded by $2^{-\rho}$ as long as

$$\mu \geq \frac{\rho + 2\log\rho + \rho\log|B| + \rho\log|\mathcal{U}|}{(|B|-\rho)\cdot(\log\ell - \log(|B|+\rho))}. \qquad (3)$$

■

As a result from the previous Claim, we get the following corollary:

Corollary 5.10 *Assuming that H is a random oracle and G is a pseudorandom generator, Protocol 6 with appropriate choice of parameters (ℓ, μ) securely computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof: We choose (ℓ, μ) as described in the proof of Claim 5.8. The proof is based on the proof of Theorem 5.2. The simulator is the same, except for the fact that it chooses the set of checks Φ as the honest sender in the real execution, sends it to the malicious receiver and receives the set of hashes $\mathcal{H}$. It then continues with the simulation as in the previous proof.

Note that the verification is passed with the exact same probability in the real and in the ideal execution. If $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{bad}}$, then the verification is passed with probability at most $2^{-\rho}$. On the other hand, if the verification is passed and $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{good}}$, then the number of consistent secrets with $(\mathcal{T}, \Phi, \mathcal{H})$ is at least 2^{κ}. Moreover, from Claim 5.8, $|\mathcal{U}| > \kappa$ and it holds also that $|\mathcal{U}| > |B|$. This implies that Claim 5.4 holds here as well. As a result, even if we give the distinguisher the bits $\mathbf{s}_B$ in the clear, there are still more than 2^{κ} possible secrets and the simulation is indistinguishable for the same reasons as previously. ■

Concrete choice of parameters. Claim 5.8 states that the bound is achieved for an appropriate choice of parameters. We numerically computed the probability in Eq. (2) 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			
$\boldsymbol{\mu}$	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

Table 3: Concrete choice of parameters for Protocol 6. $\boldsymbol{\mu}$ is computed by bounding Eq. (2) above with $2^{-\rho}$, where $|\mathcal{U}| = \kappa$, $|B| = \ell - |\mathcal{U}|$, and $\rho = 40$. Total #-checks is computed as $\mu\ell$. Each column achieves probability less than 2^{ρ} (cf. Eq. (2).)

We recall that in case we check all pairs (i.e., Protocol 5), 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.

5.4 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 given in Def. 2.1 as well as the stronger version of the assumption. Let U_ℓ denote the uniform distribution over strings of length ℓ. Another way of looking at the correlation robust function H 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 has a similar property to that of an extractor: generating highly random output from a somewhat weak random source (with sufficiently large min-entropy).

Let X be a random variable taking values from $\{0,1\}^{\ell}$. The $\mathsf{min\text{-}entropy}$ of $\mathcal{X}$, denoted $H_\infty(\mathcal{X})$, is: $H_\infty(\mathcal{X}) \stackrel{\text{def}}{=} \min_x \left\{ \log \frac{1}{\Pr[\mathcal{X}=x]} \right\} = -\log\left(\max_x \left\{ \Pr\left[\mathcal{X} = x\right] \right\}\right)$. 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 depends arbitrarily on the first random bits. We are now ready to define min-entropy correlation robustness.

Definition 5.11 (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:*

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

*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:*

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

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

In Protocol 5, the values that are used to mask the inputs of the sender are $H(j, \mathbf{t}_j), H(j, \mathbf{t}_j \oplus \mathbf{s})$ (or, $H(j, \mathbf{t}_j \oplus (s * \mathbf{e}_j)), H(j, \mathbf{t}_j \oplus (\mathbf{s} * \mathbf{e}_j) \oplus \mathbf{s})$ 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 5 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 5), 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 4 we also use the function H; however, the properties needed from H for these invocations are collision resistance (for the case of a corrupted receiver) and κ-min entropy correlation robustness (for the case of a corrupted sender). In order to emphasize the differences between the function used for the verification and the function used for the transfer phase, we denote the former by h (i.e., the one used in Step 3 of the protocol), and the latter by H (i.e., the one used in Step 4c).

Theorem 5.12

1. *Assume that H is strongly κ-min-entropy correlation robust, h is a collision resistant and κ-min-entropy correlation robust function, and G is a pseudo-random generator. Then, Protocol 4 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 and κ-min-entropy correlation robust 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.*

Proof: We prove the first item in the theorem. The second is proven in almost the same way. Moreover, we consider for now the original protocol (i.e., Protocol 5, where the checks of all pairs are performed). We later show how to consider the protocol with the reduced number of checks.

We conclude security for the corrupted sender as in Claim B.1 in Appendix B.1.

Recall that in both the ideal and real executions, the outputs of the execution consist of the randomness of the adversary, its view (the messages it receives during the execution) and the output of the honest party. The randomness of the adversary uniquely defines the messages it sends in the first round $\mathcal{T} = \{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell, \mathcal{H}_{\alpha,\beta}\}$. The view of the adversary consists of the messages it receives in the last round of the protocol, that is, $\{y_i^0, y_i^1\}_{i=1}^m$. We now consider two cases; the first in which the randomness of the adversary defines $\mathcal{T}$ for which $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$ and the second case where $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$.

$\mathcal{A}$ outputs $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$. In such a case, in both executions the adversary gets caught with probability $1 - 2^{-\rho}$. This is because both the simulator in the ideal execution and the honest sender in the real execution choose a secret $\mathbf{s}$ uniformly at random in $\{0,1\}^\ell$, and it holds that:

$$\Pr\left[\mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\right] \leq 2^{-\rho}$$

In case the verification does not pass, both the simulator in the ideal execution and the honest sender in the real execution halt the execution immediately and do not transfer the values $\{y_i^0, y_i^1\}_{i=1}^{\ell}$. As a result, the two execution are clearly identical, since the adversary has no view and the output of the honest party in both cases are $\perp$. The only possibility of failure is in case where the verification passes although $\mathcal{T} \in \mathcal{T}_{\mathsf{bad}}$, which happens with probability $2^{-\rho}$.

$\mathcal{A}$ outputs $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$. Even though $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, there is still a noticeable probability that the verification will not pass. Since the secret $\mathbf{s}$ is chosen exactly the same way in both executions, the verification passes or fails with the exact same probability.

If the verification does not pass, i.e., $\mathbf{s} \notin \mathcal{S}(\mathcal{T})$, then in both the real and the ideal executions there is no transmission, and therefore both executions are identical as above.

We left with the case where $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ and that $\mathbf{s} \in \mathcal{S}(\mathcal{T})$. In such a case, there is a transmission in both executions. We show that the two are indistinguishable by a mental experiment and consider the following three executions:

1. The real execution, conditioned on the event where $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ and $\mathbf{s} \in \mathcal{S}(\mathcal{T})$.

2. The real execution, conditioned on the event that $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ and $\mathbf{s}$ is chosen from the κ-source $\mathcal{X}(\mathcal{T})$. Below, in Claim 5.13 we show how one can sample from the set $\mathcal{S}(\mathcal{T}) = \{\mathbf{s} \in \{0,1\}^\ell \mid \mathsf{consistent}(\mathcal{T}, \mathbf{s}) = 1\}$ efficiently.

3. The ideal execution, conditioned on the event that $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$ and $\mathbf{s} \in \mathcal{S}(\mathcal{T})$.

The only difference between execution 2 and execution 3 are the values $\{y_j^{1-r_j}\}_{j=1}^m$. We recall that in the ideal execution, these values are uniform and independent, whereas in the real execution for $j = 1, \ldots, m$, it holds that $y_j^{1-r_j} = x_j^{1-r_j} \oplus H(j, \mathbf{t}'_j \oplus \mathbf{s})$. However, from the fact that H is a strongly

κ-min entropy correlation robust (as in Definition 5.11), executions 2 and 3 are computationally indistinguishable.

The only difference between execution 1 and 2 is the way $\mathbf{s}$ is chosen. In the real execution, we condition on the case where $\mathbf{s} \in \mathcal{S}(\mathcal{T})$, and thus $\mathbf{s}$ is distributed uniformly in $\mathcal{S}(\mathcal{T})$. In execution 2, $\mathbf{s}$ is chosen uniformly from the set $\mathcal{S}(\mathcal{T})$. These two executions are distributed identically.

The case of Protocol 6. We now consider the protocol with the fewer number of checks. Again, the messages $\mathcal{T} = \{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}, \mathbf{u}^1, \ldots, \mathbf{u}^\ell\}$ depends only on the randomness of $\mathcal{A}$ and therefore are the same in both executions. Both the honest sender in the real execution, and the simulator in the ideal execution, choose the functions Φ with the same distribution, and therefore the hashes $\mathcal{H} = \{\mathcal{H}_{\alpha,\phi(\alpha)}\}_{\alpha \in [\ell], \phi \in \Phi}$ have the same distribution. As the previous protocol, the case where $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{bad}}$ happens with the same probability in both executions and the view of the adversary is the same in both executions.

Given $(\mathcal{T}, \Phi, \mathcal{H})$, the case of $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{good}}$, but for which $\mathbf{s} \notin \mathcal{S}(\mathcal{T}, \Phi, \mathcal{H})$, where $\mathcal{S}(\mathcal{T}, \Phi, \mathcal{H}) = \{\mathbf{s} \in \{0,1\}^\ell \mid \mathsf{consistent}((\mathcal{T}, \Phi, \mathcal{H}), \mathbf{s}) = 1\}$ also occurs with the same probability, and the view of the adversary and the output of the honest party are clearly the same in both execution.

The case where $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{good}}$ and $\mathbf{s} \in \mathcal{S}(\mathcal{T}, \Phi, \mathcal{H})$ is handled as in the equivalent case above. Specifically, consider an execution where $\mathbf{s}$ is chosen from the source $\mathcal{X}(\mathcal{T}, \Phi, \mathcal{H})$ is defined below. This execution is identical to the real, and by the correlation robustness property of H, this execution is indistinguishable from the real, since H is a strongly κ-min entropy correlation robust. ■

Claim 5.13 *For any given transcript $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, there exists an efficient procedure that samples a uniform secret* $\mathbf{s}$ *from $S(\mathcal{T})$. This procedure is a κ-source.*

Proof: We want to show that given the transcript that the adversary has outputted, we can extract the constraints that are defined by these values, and the bits that are learned from the fact that the verification has passed. This will give us the ability to sample a value from $S(\mathcal{T})$. Note that just sampling a random $\mathbf{s} \in \{0,1\}^\ell$, and performing the same checks as in the honest execution is not enough, since there are $\{0,1\}^\ell$ possible secrets overall, whereas $|\mathcal{S}(\mathcal{T})|$ may be an order of 2^κ. As a result, the probability that a random $\mathbf{s} \in \{0,1\}^\ell$ is a consistent secret may be too small.

For a pair (α, β), consider $H_{\alpha,\beta} = (h_{\alpha,\beta}^{0,0}, h_{\alpha,\beta}^{0,1}, h_{\alpha,\beta}^{1,0}, h_{\alpha,\beta}^{1,1})$. Let $\mathsf{correct}_{p,q}(\mathcal{H}_{\alpha,\beta}) \in \{0,1\}^4$ be a predicate that its value is 1 if and only if $h_{\alpha,\beta}^{p,q} = H(G(\mathbf{k}_\alpha^p) \oplus G(\mathbf{k}_\beta^q))$. Finally, let

$$\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) \stackrel{\text{def}}{=} (\mathsf{correct}_{0,0}(\mathcal{H}_{\alpha,\beta}), \mathsf{correct}_{0,1}(\mathcal{H}_{\alpha,\beta}), \mathsf{correct}_{1,0}(\mathcal{H}_{\alpha,\beta}), \mathsf{correct}_{1,1}(\mathcal{H}_{\alpha,\beta})) \ .$$

We also assume that in cases where $\mathbf{r}^\alpha \neq \mathbf{r}^\beta$, whenever the adversary sets $h_{\alpha,\beta}^{p,q}$ that is incorrect, it sets its value to be $H(G(\mathbf{k}_\alpha^{\overline{p}}) \oplus G(\mathbf{k}_\beta^{\overline{q}}) \oplus \mathbf{u}^\alpha \oplus \mathbf{u}^\beta)$ in order to maximizes the success probability of the verification. We note that this condition can be verified as well, and generate new constraints in case it does not hold. Algorithm 1 describes how one can sample a consistent secret $\mathbf{s}$ from a given transcript $\mathcal{T}$.

ALGORITHM 1 (The κ-source $\mathcal{X}(\mathcal{T})$: sampling a uniform s from $\mathcal{S}(\mathcal{T})$)

Input: A transcript $\mathcal{T} = \{\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}, \mathbf{u}^1, \ldots, \mathbf{u}^{\ell}, \mathcal{H}_{\alpha,\beta}\}$.

1. Extract the following constraints regarding $\mathbf{s}$ and store them. For every pair (α, β) extract $\mathbf{r}^{\alpha} = \mathbf{u}^{\alpha} \oplus G(\mathbf{k}_{\alpha}^0) \oplus G(\mathbf{k}_{\alpha}^1)$ and $\mathbf{r}^{\beta}$ in a similar manner. Then:
   (a) If $\mathbf{r}^{\alpha} = \mathbf{r}^{\beta}$:
      i. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (1,1,1,1)$ then no new constraint is added.
      ii. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (1,0,0,1)$ then add the constraint $s_{\alpha} \oplus s_{\beta} = 0$.
      iii. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (0,1,1,0)$ then add the constraint $s_{\alpha} \oplus s_{\beta} = 1$.
   (b) If $\mathbf{r}^{\alpha} \neq \mathbf{r}^{\beta}$:
      i. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (1,1,0,0)$ then add the constraint $s_{\alpha} = 0$.
      ii. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (0,0,1,1)$ then add the constraint $s_{\alpha} = 1$.
      iii. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (1,0,1,0)$ then add the constraint $s_{\beta} = 0$.
      iv. If $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (0,1,0,1)$ then add the constraint $s_{\beta} = 1$.
   (c) If $\mathbf{r}^{\alpha} \neq \mathbf{r}^{\beta}$ and only one position of $H_{\alpha,\beta}$ is correct - learn both (s_{α}, s_{β}) and add this as a constraint. (e.g., if $\mathsf{correct}(\mathcal{H}_{\alpha,\beta}) = (1,0,0,0)$ then add the constraint $(s_{\alpha}, s_{\beta}) = (0,0)$).)
2. If some of the constraints are contradicting, abort and output $\perp$.
3. Otherwise, choose a random $\mathbf{s} \in \{0,1\}^{\ell}$ under the constraints that were stored above, and output it.

It is easy to see that the possible outputs of the algorithm are exactly the set $S(\mathcal{T})$. Moreover, since $\mathcal{T} \in \mathcal{T}_{\mathsf{good}}$, it holds that $|\mathcal{S}(\mathcal{T})| \geq 2^{\kappa}$. As a result, for every possible output $\mathbf{s}$ of the algorithm $\mathcal{X}(\mathcal{T})$, it holds that $\Pr[\mathcal{X}(\mathcal{T}) = \mathbf{s}] \leq 2^{-\kappa}$, and thus the min-entropy of $\mathcal{X}(\mathcal{T})$ is κ. ■

Algorithm 1 was designed for the variant of the protocol where we check all pairs. An equivalent source $\mathcal{X}(\mathcal{K}, \Phi, \mathcal{H})$ for the variant of the protocol that does not check all pairs, can also be constructed in a similar manner. The only difference between the two algorithms is that we do not run over all possible pairs (α, β) in Step 1 of the algorithm, but rather only all pairs $(\alpha, \phi(\alpha))_{\phi \in \Phi}$. This is a κ-source for every $(\mathcal{T}, \Phi) \in \mathcal{T}_{\mathsf{good}}$, since the number of possible outputs $\mathcal{S}$ is at least 2^{κ}.

5.5 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 [AL10]. 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 behaviour defines a set B with "high" cardinality. Here, 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 5, 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 5). We proceed with a formal statement of the theorem and the proof. We state and prove the theorem with respect to some concrete choice of parameters.

Theorem 5.14 *Assume that H is strongly κ-min-entropy correlation robust, h is a collision resistant and κ-min-entropy correlation robust function, and G is a pseudorandom generator. Assume the above protocol with the following parameters: ℓ is the total number of base OTs, t is the number of checks and ϵ is the deterrent factor. The protocol computes the $m \times OT_n$ functionality in the $\ell \times OT_\kappa$-hybrid model in the presence of a covert adversary with ϵ-deterrent factor, if the following conditions hold:*

1. *Let $\delta \overset{\text{def}}{=} (\ell - \kappa) \cdot \kappa - 2t(\ell - t)$. Then, we require that $\delta > 0$, and*
2. *$t > \frac{\log(1-\epsilon)}{\log\left(1-\frac{\delta}{\ell^2}\right)}$.*

Proof: The simulator is exactly the same as in Theorem 5.2, where the only difference is the consistency check. The simulator, as the sender in the real execution, chooses t random pairs $\{(\alpha_i, \beta_i)\}_{i=1}^{t}$ uniformly and independently at random. We now show that any cheating attempt gets caught with probability at least ϵ.

We again consider the graph of checks, and let $V = [\ell]$ and the edges are all possible checks. We divide $[\ell]$ to B and $\mathcal{U}$ as in the proof of Theorem 5.2, and we show that when using t checks, the probability that the adversary succeeds to pass the verification when B is too "large" is less than $1 - \epsilon$. That is, we show that for every $|B| > \ell - \kappa$ (and thus, $|\mathcal{U}| < \kappa$), the probability that the adversary passes the verification is smaller than $1 - \epsilon$.

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 no 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:

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

Recall that $\delta = (\ell - \kappa) \cdot \kappa - 2t(\ell - t)$ and that $\delta > 0$. For every $\kappa < |B| < \ell/2$ it holds that $(\ell - \kappa) \cdot \kappa < |B| \cdot |\mathcal{U}|$, and thus $2t(\ell - t) < |B| \cdot |\mathcal{U}| - \delta$ which implies that:

$$\begin{aligned} \left(\frac{1}{2} \cdot \mathsf{good} + \mathsf{bad}\right)^t &\le \left(\frac{|B|^2 + |\mathcal{U}|^2 + 2t(\ell - t) + |B| \cdot |\mathcal{U}|}{\ell^2}\right)^t \le \left(\frac{|B|^2 + |\mathcal{U}|^2 + 2|B| \cdot |\mathcal{U}| - \delta}{\ell^2}\right)^t \\ &\le \left(\frac{(|B| + |\mathcal{U}|)^2 - \delta}{\ell^2}\right)^t = \left(1 - \frac{\delta}{\ell^2}\right)^t < 1 - \epsilon \ , \end{aligned}$$

Where the last step is true since $t > \log(1 - \epsilon)/\log(1 - \frac{\delta}{\ell^2})$. ■

It is easy to verify that the statement holds for $\kappa = 128$, $\ell = 166$, $\epsilon = 0.5$ and $t = 7$. We will use these parameters in our experiments.

6 Special Purpose OT Functionalities

The protocols described up until now implement the $m \times OT_\ell$ functionality. In the following, we present further optimizations that are specifically tailored to the use of OT extensions in secure computation protocols summarized in Table 4: Correlated OT (§6.1), Sender Random OT (§6.2), Receiver Random OT (§6.3), and Random OT (§6.4). We first give the intuition and overview of the functionalities and then present a formal definitions and proofs of security.

6.1 Correlated OT (C-OT)

When performing OT extension, often the sender does not need to transfer two independent n-bit strings (x_j^0, x_j^1). In some protocols, x_j^0 and x_j^1 only need to be correlated by a value Δ_j and a correlation function f_{Δ_j}, while one of the two strings can be constant and publicly known or random. For instance, the Private Set-Intersection protocol of [DCW13] fixes $x_j^0 = 0$ and transfers only x_j^1 (hence, we can set $\Delta_j = x_j^1$ and $f_{\Delta_j}(x_j^0) = \Delta_j$) and the Hamming Distance Protocol of [BCP13] requires a random x_j^0 and a correlated $x_j^1 = f_{\Delta_j}(x_j^0) = x_j^0 + \Delta_j$. We can alter the functionality of our OT extension protocols to compute correlated OT as follows. Since x_j^0 is just a random value, P_S can set $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = f_{\Delta_j}(x_j^0)$ and can send the *single* value $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$. P_R defines its output as $H(j, \mathbf{t}_j)$ if $r_j = 0$ or as $y_j \oplus H(j, \mathbf{t}_j)$ if $r_j = 1$. For OT on n-bit strings, we thereby reduce the communication from P_S to P_R from $2n + \ell$ to $n + \ell$ per OT.

Defining the functionality. The input x_j^0 of the sender is implicitly defined by the protocol. Nevertheless, the sender may choose x_j^1 in any arbitrarily way, including as an arbitrary function of x_j^0. That is, in the protocol the sender has the freedom to choose x_j^1 as a function of x_j^0. When defining the corresponding functionality, we need to model this fact. As a result, the functionality C-OT is defined as a *reactive* functionality, where the functionality chooses x_j^0 at random, gives it to the sender, and then the sender replies with its choice for x_j^1. We proceed with a formal description of the functionality (Functionality 1), the protocol (Protocol 7) and its proof of security (Theorem 6.1).

FUNCTIONALITY 1 (The Correlated OT Functionality C-OT)

1. P_R sends its input $\mathbf{r} = (r_1, \ldots, r_m)$.
2. The functionality chooses m random n-bit strings $x_1^0, \ldots, x_m^0$ and send them to P_S.
3. P_S sends $x_1^1, \ldots, x_m^1$ to the functionality.
4. P_R gets as output $x_1^{r_1}, \ldots, x_m^{r_m}$.

PROTOCOL 7 (Implementing Correlated OT (C-OT))
We follow protocol 4, where the sender has input $f_{\Delta_1}, \ldots, f_{\Delta_m}$ instead of $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$. In Step 4c, we have the following modification:

1. P_S defines $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = f_{\Delta_j}(x_j^0)$.
2. P_S sends y_j for every $1 \le j \le m$, where: $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$.
3. For $1 \le j \le m$, P_R computes $x_j = H(j, \mathbf{t}_j)$ if $r_j = 0$, and $x_j = y_j \oplus H(j, \mathbf{t}_j)$ otherwise.

Output: P_S outputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$, P_R outputs $\mathbf{r}$.

Theorem 6.1 *Assuming that H is a programmable random oracle and G is a pseudorandom generator, then Protocol 7 (with appropriate choice of parameters, as in Claim 5.8 and Table 3) securely computes the C-OT functionality (Functionality 1) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch: We sketch the simulator and the proof, and relate to the full proof of the protocol (Theorem 5.2).

The case of corrupted Sender. The case of corrupted sender here is more subtle than the proof of the general protocol, and the functionality is now a reactive one. Moreover, we prove security in the *programmable* random oracle model.

The simulator chooses a random input $\mathbf{r}$ and follows the execution of the protocol with the corrupted sender and with an honest receiver with input $\mathbf{r}$. Specifically, the adversary first outputs a vector $\mathbf{s}$ of size ℓ. The simulator chooses random $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$ and sends them back to the adversary, together with the $\mathbf{u}^i$ messages and the necessary checks $\mathcal{H}_{\alpha,\beta}$, all set according to the protocol specifications. Note that this determines the matrices T and Q.

The simulator then receives the inputs $x_1^0, \ldots, x_m^0$ from the trusted party, and it programs the random oracle H such that for every $1 \leq j \leq m$, $H(j, \mathbf{q}_j) = x_j^0$ and chooses random output for $H(j, \mathbf{q}_j \oplus \mathbf{s})$. In case the adversary has already queried H for one of these values before the simulator programs it, the simulator is failed. The simulator receives from the adversary the messages $y_1, \ldots, y_m$, defines for every $1 \leq j \leq m$ the input $x_j^1 = y_j \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$, and sends the inputs $x_1^1, \ldots, x_m^1$ to the trusted party.

Clearly, the probability that the adversary that makes at most q queries to $H(j, \mathbf{q}_j)$ or $H(j, \mathbf{q}_j \oplus \mathbf{s})$ before it receives the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\ell$ is bounded by $q \cdot 2^{-\ell}$, and therefore the probability that the simulation fails is bounded by this amount.

The case of corrupted Receiver. The simulator is the same as in Theorem 5.2, where in the last step, instead of sending to the adversary the two messages y_j^0, y_j^1, it sends only y_j^1. Note that if no critical query (as in Definition 5.5) then the input $x_j^{1-r_j}$ is hidden from the adversary or the distinguisher. Specifically, in case $r_j = 0$, the value $\mathbf{t}_j = \mathbf{q}_j$ and therefore $H(j, \mathbf{q}_j \oplus \mathbf{s})$ is distributed uniformly, and the value $y_j = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$ is distributed uniformly as well. In case $r_j = 1$ it holds that $\mathbf{t}_j = \mathbf{q}_j \oplus \mathbf{s}$, which implies that $x_j^0 = H(j, \mathbf{q}_j) = H(j, \mathbf{t}_j \oplus \mathbf{s})$ is distributed uniformly and hidden from the adversary. ■

6.2 Sender Random OT (SR-OT)

When using OT extensions for implementing the OT-based Private Set Intersection (PSI) protocol of [PSZ14, PSSZ15], the efficiency can be improved even further. In this case, the inputs for P_S in every OT are *independent random* strings m^0 and m^1. Thus, the sender can allow the OT extension protocol (functionality) Sender Random OT (SR-OT) to determine *both* of its inputs randomly. This is achieved in the OT extension protocol by having P_S define $m^0 = H(j, \mathbf{q}_j)$ and $m^1 = H(j, \mathbf{q}_j \oplus \mathbf{s})$. Then, P_R computes m^{r_j} just as $H(j, \mathbf{t}_j)$. With this optimization, we obtain that the entire communication in the OT extension protocol consists only of the initial base-OTs, together with the messages $\mathbf{u}^1, \ldots, \mathbf{u}^\kappa$, and there are *no* y_j messages. This is a dramatic improvement of bandwidth. In particular, for the OT-PSI protocol of [PSZ14, PSSZ15], which performs $O(n\sigma)$ OTs on $\rho + 2\log_2(n)$ bit-strings, where n is the number of elements in both parties sets, σ is the bit-length of each element, and ρ is the statistical security parameter, the communication from P_S to P_R is reduced from $O(n\sigma)$ to $O(n)$.

Formal description of the functionality. We proceed with a formal description of the functionality (Functionality 2), the protocol (Protocol 8) and its proof of security (Theorem 6.2).

Theorem 6.2 *Assuming that H is a programmable random oracle, G is a pseudorandom generator, Protocol 8 (with appropriate choice of parameters, as in Claim 5.8 and Table 3) securely computes the SR-OT functionality (Functionality 2) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch: The case of corrupted Sender. We prove security in the programmable random

FUNCTIONALITY 2 (Sender Random OT)

- **Input:** P_S holds no input, P_R holds $\mathbf{r} = (r_1, \ldots, r_m)$.
- **The functionality:** The functionality chooses m pairs of random strings of size n each, $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$.
- **Output:** P_S outputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$. P_R outputs $(x_1^{r_1}, \ldots, x_m^{r_m})$.

PROTOCOL 8 (Implementing Sender Random OT (SR-OT))
We follow Protocol 4, where the sender does not have any input. In Step 4c, we have the following modification:

1. P_S defines $x_j^0 = H(j, \mathbf{q}_j)$ and $x_j^1 = H(j, \mathbf{q}_j \oplus \mathbf{s})$ for every $1 \leq j \leq m$.
2. P_R defines $x_j^{r_j} = H(j, \mathbf{t}_j)$ for every $1 \leq j \leq m$. Note that there is no interaction between the parties in this step.

Output: The sender outputs (x_j^0, x_j^1), the receiver outputs $x_j^{r_j}$.

oracle model.

The simulator chooses a random input $\mathbf{r}$ and follows the execution of the protocol with the corrupted sender and with an honest receiver with input $\mathbf{r}$. Specifically, the adversary first outputs a vector $\mathbf{s}$ of size ℓ. The simulator chooses random $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$ and sends them back to the adversary, together with the $\mathbf{u}^i$ messages and the necessary checks $\mathcal{H}_{\alpha,\beta}$, all set according to the protocol specifications. Note that this determines the matrices T and Q.

The simulator then receives the inputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$ from the trusted party, and it *programs* the random oracle H such that for every $1 \leq j \leq m$, $H(j, \mathbf{q}_j) = x_j^0$ and $H(j, \mathbf{q}_j \oplus \mathbf{s}) = x_j^1$.

The case of corrupted Receiver. The simulator is the same as in Theorem 5.2, where the only modification is that the simulator does not send the receiver any message in the transfer phase. Assuming that the receiver or the distinguisher do not make any critical query (Definition 5.5), the value $H(j, \mathbf{t}_j \oplus \mathbf{s})$ is hidden and distributed uniformly. In case where $r_j = 0$, this value is x_j^1 and in case where $r_j = 1$, it is x_j^0. The theorem follows. ∎

6.3 Receiver Random OT (RR-OT)

Analogously to the Sender Random OT, in the Receiver Random OT (RR-OT), P_R obtains his input choice bits $\mathbf{r}$ as random output of the protocol execution. Our instantiation of RR-OT in OT extension allows P_R to save one bit of communication per OT. Recall that in Step 2(a) in Protocol 4, P_R sends $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$ for $1 \leq i \leq \ell$. However, if we allow $\mathbf{r}$ to be randomly chosen, we can set $\mathbf{r} = G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$ and $\mathbf{t}^1 = G(\mathbf{k}_1^0)$ and only need to transfer $\mathbf{u}^{i'} = G(\mathbf{k}_{i'}^0) \oplus G(\mathbf{k}_{i'}^1) \oplus \mathbf{r}$ for $2 \leq i' \leq \ell$. P_S can then compute $\mathbf{q}^1 = G(\mathbf{k}_1^{s_1})$ and, as before, $\mathbf{q}^{i'} = (s_{i'} \cdot \mathbf{u}^{i'}) \oplus G(\mathbf{k}_{i'}^{s_{i'}})$. Thereby, the communication from P_R to P_S is reduced by one bit per OT.

We proceed with a formal description of the functionality (Functionality 3), protocol (Protocol 9) and its proof of security (Theorem 6.3).

Theorem 6.3 *Assuming that H is a random oracle, G is a pseudorandom generator, Protocol 9 (with appropriate choice of parameters, as in Claim 5.8 and Table 3) securely computes the RR-OT functionality (Functionality 3) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

FUNCTIONALITY 3 (The Receiver Random OT Functionality (RR-OT))

- **Input:** P_S holds m pairs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$ of n-bit strings.
- **In case of corrupted receiver:** P_R sends m-bits $\mathbf{r} = (r_1, \ldots, r_m)$ to the functionality.
- **In case of honest receiver:** P_R has not input. The functionality chooses m random bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Output:** P_S has no output; P_R outputs $(x_1^{r_1}, \ldots, x_m^{r_m})$ and $\mathbf{r}$.

PROTOCOL 9 (Implementing Receiver Random OT (RR-OT))
We follow Protocol 4 with the following modifications:

1. P_R has no input.
2. Given the chosen keys $\{\mathbf{k}_i^0, \mathbf{k}_i^1\}_{i=1}^{\ell}$, P_R sets $\mathbf{r} = G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$.
3. For every $2 \le i \le \ell$, P_R sets $\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus \mathbf{r}$, and sends $\mathbf{u}^2, \ldots, \mathbf{u}^\ell$ to P_S. Note that $\mathbf{u}^1$ is not sent.
4. In case of our actively secure OT extension protocol, the parties check consistency as previously.
5. P_R defines $T = [\mathbf{t}^1 \mid \ldots \mid \mathbf{t}^\ell]$ where $\mathbf{t}^i = G(\mathbf{k}_i^0)$ for every $1 \le i \le \ell$ as in Protocol 4.
6. P_S defines $Q = [\mathbf{q}^1 \mid \ldots \mid \mathbf{q}^\ell]$ where $\mathbf{q}^1 = G(\mathbf{k}_1^{s_1})$, and for every $2 \le i \le \ell$, $\mathbf{q}^i$ is defined as in Protocol 4, i.e., $\mathbf{q}^i = G(\mathbf{k}_i^0)$ if $s_i = 0$; otherwise, set $\mathbf{q}^i = \mathbf{u}^i \oplus G(\mathbf{k}_i^1)$.
7. The parties proceed with the execution as in Protocol 4.

Proof Sketch: Note that the random oracle does not have to be programmable. Regarding correctness, for every $2 \le i \le \ell$ it holds that $\mathbf{q}^i = \mathbf{t}^i \oplus (s_i \cdot \mathbf{r})$. For $i = 1$, if $s_1 = 0$ then $\mathbf{q}^1 = G(\mathbf{k}_1^0) = \mathbf{t}^i$; in case $s_1 = 1$ then $\mathbf{q}^1 = G(\mathbf{k}_1^1) = G(\mathbf{k}_1^0) \oplus \mathbf{r} = \mathbf{t}^i \oplus \mathbf{r}$, and therefore $\mathbf{q}^1 = \mathbf{t}^1 \oplus (s_1 \cdot \mathbf{r})$ as well.

The case of corrupted sender. The simulator is exactly the same as in Theorem 5.2, i.e., the simulator chooses an random $\mathbf{r}'$ and plays the role of an honest receiver with input $\mathbf{r}'$. There is no contradiction between the simulated execution (where the input of the receiver is $\mathbf{r}'$) and the actual value $\mathbf{r}$ chosen by the trusted party, for the same reasons that the simulator in Theorem 5.2 succeeds with the simulation for some random input $\mathbf{r}'$ whereas the receiver uses its true input $\mathbf{r}$ to the trusted party.

The case of corrupted receiver. The only difference is that the simulator sends the messages $\mathbf{u}^2, \ldots, \mathbf{u}^\ell$ (excluding $\mathbf{u}^1$). In particular, the input $\mathbf{r}$ that the simulator extracts is the most repeated $\mathbf{r}^i$ value according to the messages $\mathbf{u}^2, \ldots, \mathbf{u}^\ell$, and define $\mathbf{r}^1$ as $G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1)$. The theorem follows from the correctness argument as above, and Theorem 5.2. ■

6.4 Random OT (R-OT)

In a random OT, both P_S and P_R obtain their input as random output of the protocol. The random OT functionality can be obtained by combining the SR-OT protocol with the RR-OT protocol. Random OT can be used in the GMW protocol when pre-computing random multiplication triples (see §2.7). We proceed with a formal description of the functionality (Functionality 4), the protocol (Protocol 10) and its proof of security (Theorem 6.4).

FUNCTIONALITY 4 (Functionality Random OT R-OT)

- **Inputs:** P_S has no input, and the functionality chooses m pairs of n-bit strings $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$.
- **In case of corrupted receiver:** P_R sends m-bits $\mathbf{r} = (r_1, \ldots, r_m)$ to the functionality.
- **In case of honest receiver:** P_R has no input. The functionality chooses m random bits $\mathbf{r} = (r_1, \ldots, r_m)$.
- **Output:** P_S outputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$. P_R outputs $(x_1^{r_1}, \ldots, x_m^{r_m})$ and $\mathbf{r}$.

PROTOCOL 10 (Implementing Random-OT R-OT)
This is a simple combination of Protocols 8 and 9. Specifically, P_R defines its input as $G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$, and P_S defines its inputs x_j^0, x_j^1 according to $H(j, \mathbf{q}_j), H(j, \mathbf{q}_j \oplus \mathbf{s})$, respectively, for every $1 \leq j \leq m$. There is no transmission of $\mathbf{u}^1$ from P_R to P_S, and there is no transmission of y_j^0, y_j^1 from P_S to P_R for every $1 \leq j \leq m$.

Theorem 6.4 *Assuming that H is a programmable random oracle, G is a pseudorandom generator, Protocol 10 (with appropriate choice of parameters, as in Claim 5.8 and Table 3) securely computes the R-OT functionality (Functionality 4) in the $\ell \times OT_\kappa$-hybrid model in the presence of a static malicious adversary.*

Proof Sketch: The proof follows from Theorems 6.3 and 6.2. In particular, in case of corrupted sender the simulator receives the inputs $(x_1^0, x_1^1), \ldots, (x_m^0, x_m^1)$ from the trusted party, and it *programs* the random oracle H such that for every $1 \leq j \leq m$, $H(j, \mathbf{q}_j) = x_j^0$ and $H(j, \mathbf{q}_j \oplus \mathbf{s}) = x_j^1$. In case of a corrupted receiver, the input $\mathbf{r}$ that the simulator extracts and sends to the trusted party is the most repeated $\mathbf{r}^i$ value according to the messages $\mathbf{u}^2, \ldots, \mathbf{u}^\ell$ (where $\mathbf{r}^1$ is defined as $G(\mathbf{k}_1^0) \oplus G(\mathbf{k}_1^1)$). ■

Summary. The original OT extension protocol of [IKNP03] and our proposed improvements for $m \times OT_n$ are summarized in Tab. 4. We compare the communication complexity of P_R and P_S for m parallel 1-out-of-2 OT extensions of n-bit strings, with security parameter κ and ℓ base-OTs (we omit the cost of the initial $\kappa \times OT_\kappa$). We also compare the assumption on the function H needed in each protocol, where CR denotes Correlation Robustness and RO denotes Random Oracle.

Protocol	**Applicability**	$P_R \to P_S$	$P_S \to P_R$	H
Original [EGL85]+[IKNP03]	All applications	$m\ell$	$2mn$	CR
C-OT §6.1	x_j^0 random; x_j^1 correlated with Δ_j	$m\ell$	mn	RO
SR-OT §6.2	x_j^0, x_j^1 random, r_j chosen	$m\ell$	0	RO
RR-OT §6.3	x_j^0, x_j^1 chosen, r_j random	$m(\ell-1)$	$2mn$	RO
R-OT §6.4	x_j^0, x_j^1, r_j random	$m(\ell-1)$	0	RO

Table 4: Bits sent for sender P_S and receiver P_R for m 1-out-of-2 OT extensions of n-bit strings and security parameter κ for the semi-honest OT extension protocol of [IKNP03] with our optimizations.

7 Experimental Evaluation

In this section, we empirically evaluate our optimizations and proposed protocols. In §7.1 we describe our benchmarking environment and implementation. In §7.2 we evaluate the optimizations on the passively secure OT extension protocol of [IKNP03], outlined in §4. In §7.3, we evaluate the special purpose OT functionalities, presented in §6. In §7.4, we evaluate the performance of our covert and actively secure OT extension protocol, presented in §5. Finally, in §7.5, we evaluate the overhead for the min-entropy correlation robustness assumption from §5.4 compared to the random oracle assumption.

7.1 Benchmark Setting

Parameters and Instantiation. In all our experiments, we assume long-term security (cf. Table 2), i.e., we set the symmetric security parameter $\kappa = 128$-bit and use the 283-bit Koblitz curve of [NIS12]. We instantiate the pseudorandom generator using AES-CTR and the correlation-robust function as well as the random oracle using SHA256. We process the OTs blockwise with blocks of size $w = 2^{19}$. We use the OT protocol of [NP01] in the random oracle model as base-OT protocol for the passively secure OT extension protocols and the OT protocol of [CO15] as base-OT protocol for the actively secure OT extension protocols. As parameters for our actively secure protocol we use 190 base-OTs and 380 checks and for our covert secure protocol we use 166 base-OTs and 7 checks as these parameters resulted in the best performance. For the actively secure protocol of [NNOB12], we use the parameters in the paper, i.e. 342 base-OTs and 171 checks. Our implementation is available online at `http://encrypto.de/code/OTExtension`.

3-step OT Extension. To generate large numbers $m > w = 2^{19}$ of OTs for the actively secure OT extension protocols, we perform a 3-step OT, where P_S and P_R first perform ℓ base-OTs, then extend these ℓ OTs to $\lceil m\ell/w \rceil$ OTs using the respective OT extension protocols, and finally split these $\lceil m\ell/w \rceil$ OTs into m/w blocks of ℓ OTs and extend each block to w OTs again using the respective OT extension protocol to obtain the m OTs. In case $m > w\lfloor w/\ell \rfloor$, one can simply extend this approach again and do a 4-step OT.

1-out-of-2 R-OT on bits via 1-out-of-N OT [KK13]. For the passively secure 1-out-of-N OT extension protocol of [KK13], we use $N = 16$, since this resulted in the lowest communication, and hence convert one 1-out-of-16 OT to four 1-out-of-2 OTs. In particular, we compute the SR-OT functionality and convert the i-th 1-out-of-16 OT with 4-bit outputs values $(z_i^0, ..., z_i^{15}) \in \{0,1\}^{64}$ to the $4i$-th to $(4i+3)$-th 1-out-of-2 OTs with single bit output values $(x_{4i}^0, x_{4i}^1), ..., (x_{4i+3}^0, x_{4i+3}^1)$ as: $(x_{4i}^0 || x_{4i+1}^0 || x_{4i+2}^0 || x_{4i+3}^0) = z_i^0$ and $(x_{4i}^1 || x_{4i+1}^1 || x_{4i+2}^1 || x_{4i+3}^1) = z_i^{15}$. For the remaining values $(z_i^1, ..., z_i^{14})$, P_S sends 4-bit correction values $y_i^j = z_i^j \oplus (x_{4i}^{j_0} || x_{4i+1}^{j_1} || x_{4i+2}^{j_2} || x_{4i+3}^{j_3})$ for $1 \leq j \leq 14$, $j = j_0 || j_1 || j_2 || j_3$ and j_0 is the least significant bit of j. Thereby, we do not need to send the correction values for z_i^0 and z_i^{15} which saves 8-bits of communication per 1-out-of-16 OT. To compute the RR-OT functionality, P_R randomly selects four bit positions that uniquely determine the codeword in the base-OTs and omits the sending of the correction values $\mathbf{u}$ for these positions. Note, that the [KK13] OT can also be instantiated with $N \in \{2, 4, 8\}$, which would increase communication but reduce computation complexity. As a special case, if $N = 2$ and when using a repetition code, the [KK13] protocol would be equal to the [IKNP03] protocol.

Benchmark Environment. We perform our experiments in two settings: a *LAN* setting and a *WAN* setting. In the LAN setting, we run the sender and receiver routines on two Desktop

PCs, each equipped with an Intel Haswell i7-4770K CPU with 4 cores and AES-NI support and 16 GB RAM that are connected by Gigabit Ethernet. In the WAN setting, we run the sender on an Amazon EC2 m3.xlarge instance with a 2.5 GHz Intel Xeon E5-2670v2 CPU with 4 virtual CPUs (vCPUs) and 15 GB RAM, located in North Virginia (US EAST) and the receiver routine on one of our Desktop PCs in Europe. The average bandwidth between these two machines was 120MBit/s and the average ping latency (round-trip time) was 100 ms.

7.2 Evaluation of Semi-Honest OT Extension

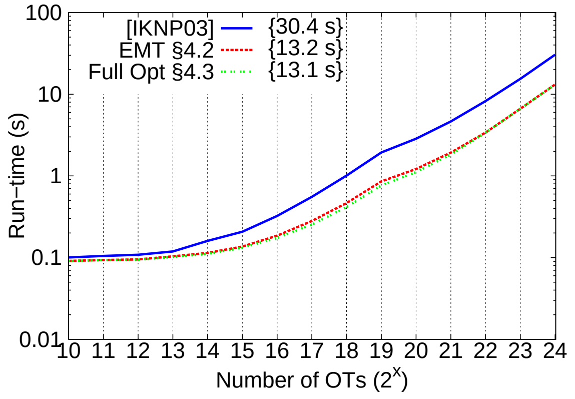
In the following, we evaluate the performance gains from our optimizations on the passively secure OT extension protocol of [IKNP03] described in §4. We benchmark the protocol in three versions: the original passively secure OT extension protocol of [IKNP03] with naive matrix transposition, the protocol of [IKNP03] with the Eklundh matrix transposition (cf. §4.2), and our improved passively secure OT extension protocol (cf. §4.3), including the Eklundh matrix transposition. We evaluate all three versions using the Random OT functionality (cf. §6.4) as this functionality reduces the overhead for the last step in the protocol and hence lets us evaluate the core-efficiency of the protocol more precisely. We vary the number of OTs from 2^{10} (=1,024) to 2^{24} (=16,777,216) and fix the bit-length of the transferred strings to 128. The results in the LAN and WAN setting are given in Figure 2.

In both the LAN and WAN setting, we were able to decrease the run-time by factor 2-3. In the LAN setting, the efficient matrix transposition from §4.2 had the highest impact while our protocol optimization from §4.3 only slightly decreased the run-time. This can be explained by the computation being the bottleneck in the LAN setting, hence the communication improvement from our protocol optimization had only little effect. In the WAN setting, on the other hand, the communication improvement from our protocol optimization resulted in a higher run-time decrease than the efficient matrix transposition because in this setting communication is the main bottleneck. For smaller number of OTs, the base-OTs have a high impact on the total run-time, hence the run-time of all protocols is similar. However, the base-OTs amortize for higher number of OTs and the improvements on the OT extension protocols can be seen more clearly. Note that the dent for 2^{19} OTs for both the LAN and WAN setting is due to the block size of our implementation. More than 2^{19} OTs are processed in multiple blocks, resulting in a better amortization.

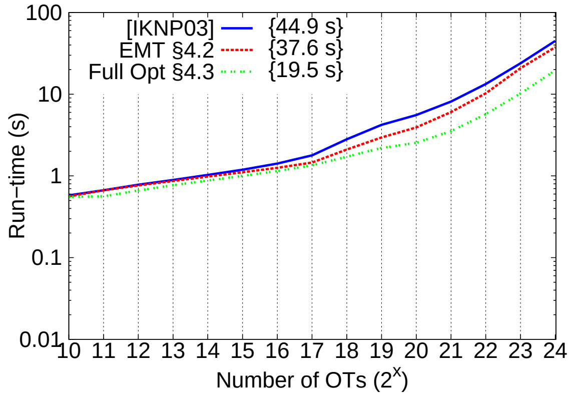
7.3 Evaluation of Special Purpose OT Functionalities

Next we evaluate the performance of the special purpose OT functionalities, outlined in §6. We use the performance of the Random OT (R-OT) (cf. §6.4) as base-line and evaluate the overhead that is added when using the the original OT, Correlated OT (C-OT) (cf. §6.1), and Sender Random OT (SR-OT) (cf. §6.2) functionalities. Simialar to the evaluation of semi-honest protocol optimizations, we vary the number of OTs from 2^{10} (=1,024) to 2^{24} (=16,777,216) and fix the bit-length of the transferred strings to 128. The results for the LAN and WAN scenario are given in Figure 3.

From the results we can observe that the standard OT functionality and the C-OT functionality are both slower than the R-OT functionality. The SR-OT, on the other hand, has a similar performance as the R-OT since R-OT only reduces the communication by a single bit per OT. In the LAN setting, the performance difference is nearly negligible (2^{24} R-OTs need 13.1 s while the same number of OTs require 13.6 s), since the improvements from R-OT mainly affect the communication complexity which is not the bottleneck in the LAN setting. In the WAN setting,



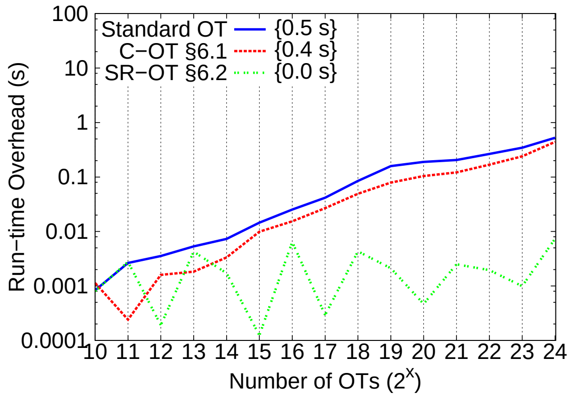
(a) LAN Setting



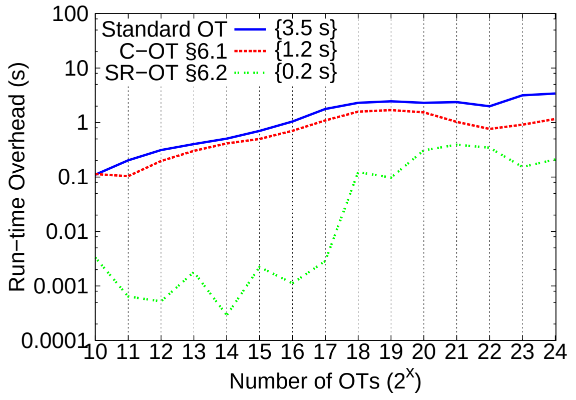
(b) WAN Setting

Figure 2: Run-time for passively secure R-OT extension on 128-bit strings in the LAN (a)- and WAN (b) setting. Time for 2^{24} OTs given in {}.

however, the performance improvements of (S)R-OT are higher, since the communication is the bottleneck and the C-OT and standard OT functionality have to send messages from the sender to the receiver. Evaluating 2^{24} OTs in the WAN setting requires 23.0 s for the standard OT functionality, 20.7 s for the C-OT functionality, 19.7 s for SR-OT, and 19.5 s for R-OT.

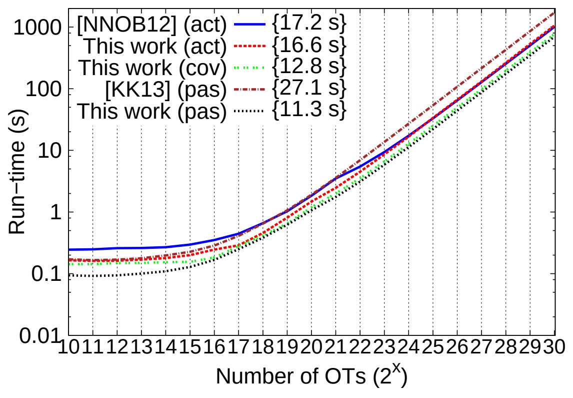


(a) LAN Setting

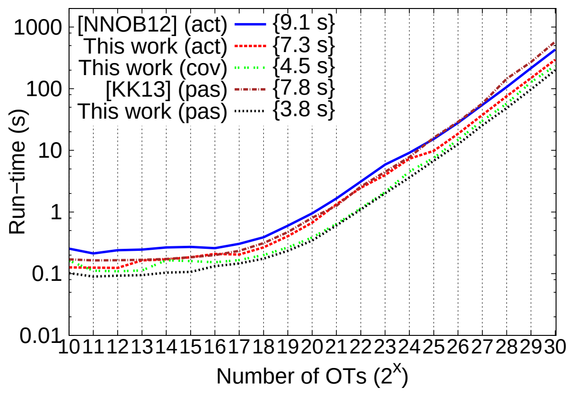


(b) WAN Setting

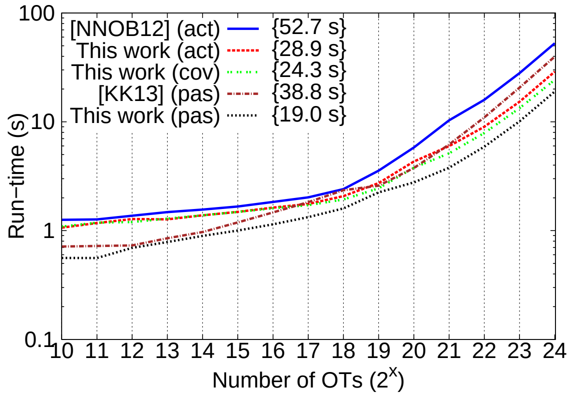
Figure 3: Run-time overhead over R-OT for different OT flavors using the semi-honest OT extension on 128-bit strings in the LAN (a)- and WAN (b) setting. Run-time overhead for 2^{24} OTs given in {}.



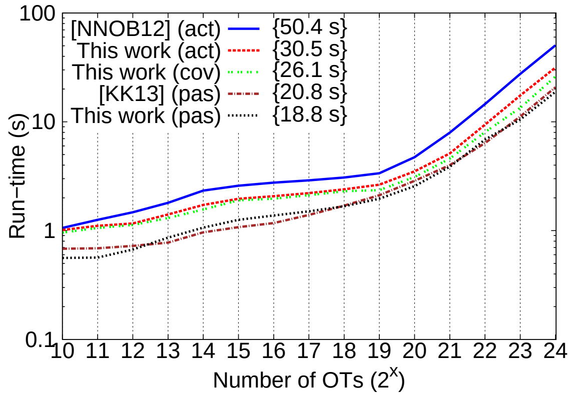
(a) LAN Setting Single Thread



(b) LAN Setting Four Threads



(c) WAN Setting Single Thread



(d) WAN Setting Four Threads

Figure 4: Run-time for single thread (a,c) and multi thread (b,d) passively, covert, and actively secure R-OT extension protocols on 1-bit strings in the LAN (a,b)- and WAN (c,d) setting. Time for 2^{24} OTs given in {}.

7.4 Evaluation of Actively Secure OT Extension

Here we evaluate our covert- and actively secure OT extension protocols of §5 and compare their performance to the actively secure protocol of [NNOB12], the passively secure 1-out-of-N OT protocol of [KK13], and our optimized version of the passively secure protocol of [IKNP03]. We benchmark all five protocols on the 1-bit Random OT functionality and vary the number of OTs from 2^{10} (=1,024) to 2^{30} (=1,073,741,824) in the LAN setting and to 2^{24} (=16,777,216) in the WAN setting. We evaluate the protocols once using one thread for both parties and once using four threads for both parties to highlight the effect of increased computing power. The single- and multi-thread results are given in Figure 4. To evaluate the improvement when using multiple threads in parallel, we benchmark all protocols in the LAN setting on a fixed number of 2^{26} random OTs with increasing number of threads from 1 to 4 and give the results in Table 5.

Single Thread. As expected, we can observe that the run-time increases with the provided security as our passively secure OT extension protocol outperforms our covert secure protocol which again outperforms both actively secure protocols in both LAN and WAN. The only exception to this is the passively secure 1-out-of-N OT extension of [KK13], which is slowest in the LAN setting and second slowest in the WAN setting due to its higher computational overhead. In the LAN setting, the actively secure protocol of [NNOB12] outperforms our actively secure protocol since our protocol has a larger computational overhead for the check routine. In the WAN setting, however, the communication becomes the bottleneck and the overhead for the communication of [NNOB12] outweighs the computational overhead for the check routine of our protocol. In fact, in the WAN setting, the run-time overhead of the covert- and actively secure OT extension protocols over the passively secure protocol is proportional to their communication overhead. Our covert secure protocol has a communication and run-time overhead of 130%, our actively secure protocol has a communication overhead of 148% and a run-time overhead of 152%, and the actively secure protocol of [NNOB12] has a communication overhead of 267% and a run-time overhead of 277%.

Multiple Threads. The main improvement when increasing the number of threads can be seen in the LAN setting, were the run-time of all protocols was improved. In particular the passively secure OT extension protocol of [KK13] and our actively secure protocol benefit most from the increased number of threads (cf. Table 5). The better scaling of these protocols can again be explained by their lower communication, which becomes the bottleneck when using multiple threads even in the LAN setting. In the WAN setting, the run-times for nearly all protocols remain unchanged even when using multiple threads since already a single thread is able to utilize the full bandwidth. The only exception to this is the passively secure protocol of [KK13], which nearly achieves the same run-times as our passively secure protocol.

Protocol	**1 Thread**	**2 Threads**	**3 Threads**	**4 Threads**	**Improvement** $1 \mapsto 4$
[NNOB12] (act)	65.8 s	34.6 s	27.2 s	27.3 s	2.4×
This work (act)	64.7 s	33.1 s	23.7 s	18.8 s	3.4×
This work (cov)	49.9 s	25.8 s	18.2 s	15.1 s	3.3×
This work (pas)	44.1 s	22.7 s	15.9 s	13.2 s	3.3×
[KK13] (pas)	107.7 s	54.6 s	37.4 s	29.5 s	3.7×

Table 5: Run-time for increasing number of threads and time improvement of 4 threads over 1 thread when evaluating 2^{26} random OT extensions on 1-bit strings in the LAN setting.

7.5 Evaluation of Min-Entropy Correlation Robustness

We empirically evaluate the overhead when using the min-entropy correlation robust (MECR) version (cf. §5.4) instead of the random oracle (RO) version (cf. §5.1) of our actively secure OT extension protocol. Recall, that we achieve the min-entropy correlation robustness by changing Step 4c in Protocol 5 such that the sender chooses random $\mathbf{d}_j \in \{0,1\}^\ell$ and computes $y_j^0 = x_j^0 \oplus H(j, \mathbf{q}_j \oplus \mathbf{d}_j)$ and $y_j^1 = x_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{d}_j \oplus \mathbf{s})$. The sender then sends $(\mathbf{d}_j, y_j^0, y_j^1)$ to the receiver who computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}_j \oplus \mathbf{d}_j)$. We benchmark the protocol on 2^{10} to 2^{24} actively secure random OTs on 1-bit strings in the LAN setting and give the overhead of the MECR version over the RO version in Figure 5.

From the results we can observe that the MECR version adds a constant overhead per block of OTs. While this overhead remains constant and low for less than 2^{19} OTs, it grows linearly in the number of OTs that are processed. For 2^{24} OTs, the difference amounts to 1.4 s, where the MECR version has a run-time of 30.3 s, while the RO version has a run-time of 28.9 s.

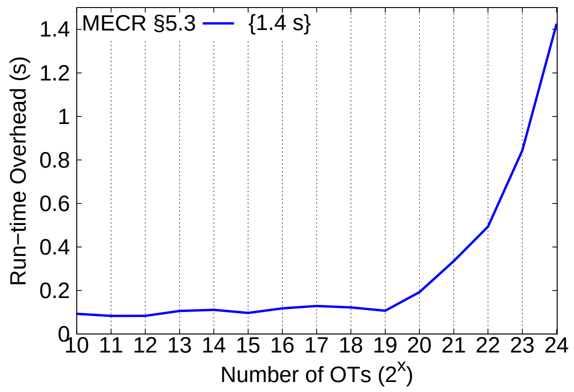


Figure 5: Run-time overhead of the min-entropy correlation robustness (MECR) version of our actively secure OT extension protocol compared to the random oracle version using random OT on 1-bit strings in the LAN setting. Time overhead for 2^{24} OTs given in {}.

Acknowledgements

This work is 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 author 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 German Federal Ministry of Education and Research (BMBF) within CRISP, by the DFG as part of project E4 within the CRC 1119 CROSSING, and by the Hessian LOEWE excellence initiative within CASED. We would like to thank the anonymous reviewers of the Journal of Cryptology for their valuable comments on our work. Moreover, we thank Peter Scholl for pointing out a security flaw for the case of a corrupted malicious sender in an earlier version of this paper.

References

[AL10] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Journal of Cryptology*, volume 23(2), pages 281–343. Springer, 2010.

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Computer and Communications Security (CCS'13)*, pages 535–548. ACM, 2013. Code: `http://encrypto.de/code/OTExtension`.

[ALSZ15] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *LNCS*, pages 673–701. Springer, 2015. Full version: `http://eprint.iacr.org/2015/061`.

[BCP13] J. Bringer, H. Chabanne, and A. Patey. SHADE: secure hamming distance computation from oblivious transfer. In *Financial Cryptography and Data Security (FC'13)*, volume 7862 of *LNCS*, pages 164–176. Springer, 2013.

[Bea91] D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.

[Bea96] D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Symposium on the Theory of Computing (STOC'96)*, pages 479–488. ACM, 1996.

[BHKR13] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE, 2013.

[BLN$^+$15] S. S. Burra, E. Larraia, J. B. Nielsen, P. S. Nordholt, C. Orlandi, E. Orsini, P. Scholl, and N. P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. *IACR Cryptology ePrint Archive*, 2015:472, 2015.

[BNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Computer and Communications Security (CCS'08)*, pages 257–266. ACM, 2008.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CHK$^+$12] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein. Secure multi-party computation of Boolean circuits with applications to privacy in on-line marketplaces. In *Cryptographers' Track at the RSA Conference (CT-RSA'12)*, volume 7178 of *LNCS*, pages 416–432. Springer, 2012.

[CO15] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology – LATINCRYPT'15*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.

[DCW13] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: An efficient and scalable protocol. In *ACM Computer and Communications Security (CCS'13)*, pages 789–800. ACM, 2013.

[DLT14] I. Damgård, R. Lauritsen, and T. Toft. An empirical study and some improvements of the MiniMac protocol for secure computation. In *Security and Cryptography for Networks (SCN'14)*, volume 8642 of *LNCS*, pages 398–415. Springer, 2014.

[DSZ15] D. Demmler, T. Schneider, and M. Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security (NDSS'15)*. The Internet Society, 2015.

[DZ13] I. Damgård and S. Zakarias. Constant-overhead secure computation of Boolean circuits using preprocessing. In *Theory of Cryptography Conference (TCC'13)*, volume 7785 of *LNCS*, pages 621–641. Springer, 2013.

[EFG$^+$09] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies Symposium (PETS'09)*, volume 5672 of *LNCS*, pages 235–253. Springer, 2009.

[EFLL12] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell. SCAPI: the secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012. Online: `http://eprint.iacr.org/2012/629`.

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *Communications of the ACM*, volume 28(6), pages 637–647. ACM, 1985.

[Ekl72] J. O. Eklundh. A fast computer method for matrix transposing. In *IEEE Transactions on Computers*, volume C-21(7), pages 801–803. IEEE, 1972.

[FAZ05] K. Frikken, M. Atallah, and C. Zhang. Privacy-preserving credit checking. In *Electronic Commerce (EC'05)*, pages 147–154. ACM, 2005.

[FKOS15] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A unified approach to MPC with preprocessing using OT. In *Advances in Cryptology – ASIACRYPT'15*, volume 9452 of *LNCS*, pages 711–735. Springer, 2015.

[FN13] T. K. Frederiksen and J. B. Nielsen. Fast and maliciously secure two-party computation using the GPU. In *Applied Cryptography and Network Security (ACNS'13)*, volume 7954 of *LNCS*, pages 339–356. Springer, 2013.

[GKK$^+$12] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM Computer and Communications Security (CCS'12)*, pages 513–524. ACM, 2012.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Symposium on Theory of Computing (STOC'87)*, pages 218–229. ACM, 1987.

[Gol04] O. Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004.

[HCE11] Y. Huang, P. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *Hot topics in security (HotSec'11)*. USENIX, 2011.

[HEK12] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security (NDSS'12)*. The Internet Society, 2012.

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security'11*, pages 539–554. USENIX, 2011.

[HFKV12] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ANSI C. In *ACM Computer and Communications Security (CCS'12)*, pages 772–783. ACM, 2012.

[HIKN08] D. Harnik, Y. Ishai, E. Kushilevitz, and J. Buus Nielsen. OT-combiners via secure computation. In *Theory of Cryptography Conference (TCC'08)*, volume 4948 of *LNCS*, pages 393–411. Springer, 2008.

[HKS$^+$10] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *ACM Computer and Communications Security (CCS'10)*, pages 451–462. ACM, 2010.

[HMEK11] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *Network and Distributed Security Symposium (NDSS'11)*. The Internet Society, 2011.

[HS13] W. Henecka and T. Schneider. Faster secure two-party computation with less memory. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'13)*, pages 437–446. ACM, 2013.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[IKOS08] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography with constant computational overhead. In *ACM Symposium on Theory of Computing (STOC'08)*, pages 433–442. ACM, 2008.

[IPS08] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.

[IR88] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Advances in Cryptology – CRYPTO'88*, volume 403 of *LNCS*, pages 8–26. Springer, 1988.

[Ker11] F. Kerschbaum. Automatically optimizing secure computation. In *ACM Computer and Communications Security (CCS'11)*, pages 703–714. ACM, 2011.

[KK13] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *Advances in Cryptology – CRYPTO'13*, volume 8043 of *LNCS*, pages 54–70. Springer, 2013.

[KOS15] M. Keller, E. Orsini, and P. Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology - CRYPTO'15*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[KSS12] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security'12*, pages 285–300. USENIX, 2012.

[KSS13] M. Keller, P. Scholl, and N. P. Smart. An architecture for practical actively secure MPC with dishonest majority. In *ACM Computer and Communications Security (CCS'13)*, pages 549–560. ACM, 2013.

[Lar14] E. Larraia. Extending oblivious transfer efficiently, or - how to get active security with constant cryptographic overhead. In *Progress in Cryptology – LATINCRYPT'14*, volume 8895 of *LNCS*, pages 368–386. Springer, 2014.

[LOS14] E. Larraia, E. Orsini, and N. P. Smart. Dishonest majority multi-party computation for binary circuits. In *Advances in Cryptology – CRYPTO'14*, volume 8617 of *LNCS*, pages 495–512. Springer, 2014.

[LP86] L. Lovász and M.D. Plummer. *Matching Theory.* Akadémiai Kiadó, Budapest, 1986. Also published as Vol. 121 of the North-Holland Mathematics Studies, North-Holland Publishing, Amsterdam.

[LP11] Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *Theory of Cryptography Conference (TCC'11)*, volume 6597 of *LNCS*, pages 329–346. Springer, 2011.

[LR15] Y. Lindell and B. Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *ACM Computer and Communications Security (CCS'15)*, pages 579–590. ACM, 2015.

[LZ13] Y. Lindell and H. Zarosim. On the feasibility of extending oblivious transfer. In *Theory of Cryptography Conference (TCC'13)*, volume 7785 of *LNCS*, pages 519–538. Springer, 2013.

[Mal11] L. Malka. VMCrypt - modular software architecture for scalable secure computation. In *ACM Computer and Communications Security (CCS'11)*, pages 715–724. ACM, 2011.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security'04*, pages 287–302. USENIX, 2004.

[MOR03] P. MacKenzie, A. Oprea, and M. K. Reiter. Automatic generation of two-party computations. In *ACM Computer and Communications Security (CCS'03)*, pages 210–219. ACM, 2003.

[Nie07] J. B. Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. Online: `http://eprint.iacr.org/2007/215`.

[NIS12] NIST. NIST Special Publication 800-57, Recommendation for Key Management Part 1: General (Rev. 3). Technical report, NIST, 2012.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology – CRYPTO'12*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.

[NP01] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Symposium on Discrete Algorithms (SODA'01)*, pages 448–457. ACM/SIAM, 2001.

[NWI$^+$13] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 334–348. IEEE, 2013.

[PSSZ15] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security'15*, pages 515–530. USENIX, 2015.

[PSZ14] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on ot extension. In *USENIX Security'14*, pages 797–812. USENIX, 2014.

[PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.

[Rab81] M. O. Rabin. *How to exchange secrets with oblivious transfer*, TR-81 edition, 1981. Aiken Computation Lab, Harvard University.

[Sch17] P. Scholl. Private correspondence, Oct 2017.

[SK11] A. Schröpfer and F. Kerschbaum. Demo: secure computation in JavaScript. In *ACM Computer and Communications Security (CCS'11)*, pages 849–852. ACM, 2011.

[SZ13] T. Schneider and M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security (FC'13)*, volume 7859 of *LNCS*, pages 275–292. Springer, 2013.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

A Active Secure OT Extension of [NNOB12]

In Protocol 11 we depict the actively-secure OT extension protocol of [NNOB12] with optimizations from [FN13].

PROTOCOL 11 (Active secure OT extension protocol of [NNOB12])

- **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 number of base-OTs $\ell = \lceil \frac{8}{3}\kappa \rceil$.
- **Oracles and cryptographic primitives:** The parties have an oracle access to the $\ell \times OT_\kappa$ functionality, and use a pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a random oracle H (see §5.4 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:*
   (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 $\mathbf{q}^i = (s_i \cdot \mathbf{u}^i) \oplus G(\mathbf{k}_i^{s_i})$. $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$.)
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 $\Pi(\pi) = \{i | i \leq \pi(i)\}$.
   (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 $1 \leq k \leq \lfloor \ell/2 \rfloor$, 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 (continued):*
   (a) Let $Q' = [\mathbf{q}'^1 | \ldots | \mathbf{q}'^{\lfloor \ell/2 \rfloor}]$ 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 $1 \leq j \leq m$, 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 $1 \leq j \leq m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, \mathbf{t}'_j)$.
5. **Output:** P_R outputs $(x_1, \ldots, x_m)$; P_S has no output.

B Security in the Presence of a Malicious Sender

We prove that our protocols are secure in the presence of a malicious Sender. We have:

Claim B.1 *Protocol 5 is secure in the presence of a malicious sender, assuming that H is a κ-min entropy correlation robust function.*

Proof: We start with the description of the simulator $\mathcal{S}$.

The simulator $\boldsymbol{\mathcal{S}}$.

- *Upon receiving auxiliary input z, invoke the adversary $\mathcal{A}$ (controlling the corrupted sender) on z.*

- *The simulator fixes $\mathbf{r} = 0^m$. It then simulates an honest execution of a receiver with input $\mathbf{r}' = 0^m || \boldsymbol{\tau}$ with the adversary $\mathcal{A}$ for a random $\boldsymbol{\tau}$. In particular, it chooses the random keys $\mathbf{k}_i^0, \mathbf{k}_i^1$. Then, it simulates the base OTs functionality for the adversary, and upon receiving the string $\mathbf{s} = (s_1, \ldots, s_\ell)$ it sends it $\{\mathbf{k}_i^{s_i}\}_{i=1}^{\ell}$. It then sends*

$$\mathbf{u}^i = G(\mathbf{k}_i^0) \oplus G(\mathbf{k}_i^1) \oplus (0^m || \boldsymbol{\tau})$$

  for every $i = 1, \ldots, \ell$. Finally, for every pair $(\alpha, \beta) \in [\ell]^2$, it computes

$$\begin{aligned} h_{\alpha,\beta}^{s_\alpha,s_\beta} &= H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta})) \ , \\ h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}} &= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}})) = H(\mathbf{u}^i \oplus \mathbf{u}^j \oplus G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta})) \end{aligned}$$

  and sets $h_{\alpha,\beta}^{s_\alpha,\overline{s_\beta}}$ and $h_{\alpha,\beta}^{\overline{s_\alpha},s_\beta}$ uniformly at random of the appropriate length.

- *When the adversary sends during the transfer phase the values (y_j^0, y_j^1) for every $j = 1, \ldots, m$, the simulator extracts the inputs (x_j^0, x_j^1) using the function H, the matrix T and the string $\mathbf{s}$. It then sends these inputs to the trusted parties as the input of the corrupted sender.*

- *The simulator outputs whatever $\mathcal{A}$ outputs, and halts.*

We now show that no distinguisher succeeds to distinguish between the output of the ideal execution and the real execution. Essentially, the key idea is to show that for every possible input $\mathbf{r}$, no distinguisher can distinguish between an honest execution of $\mathbf{r}' = \mathbf{r}||\boldsymbol{\tau}$ and $\mathbf{r}'' = 0^m||\boldsymbol{\tau}$, and that the values $h_{\alpha,\beta}^{s_\alpha,\overline{s_\beta}}$ and $h_{\alpha,\beta}^{\overline{s_\alpha},s_\beta}$ are distributed uniformly.

We now show that the joint distribution of the outputs of the parties is indistinguishable in the real world and in the ideal world. Towards this end, consider the following sequence of hybrid games:

- $\mathsf{hyb_0}$: This is the real execution (in the base OT hybrid model) with the true input $\mathbf{r}'$. The output of the execution is the output of the adversary (without loss of generality, its view) and the output of the honest receiver.

- $\mathsf{hyb_1}$**:** In this execution, we have a trusted party for computing the output of the receiver. Specifically, let $\mathbf{r}'$ be the input of the honest receiver, we run an execution of the real protocol with the adversary $\mathcal{A}$ and the receiver with input $\mathbf{r}'$. When the adversary sends the messages $\{(y_j^0, y_j^1)\}_{j=1}^m$ in the last step of the protocol, we use the string $\mathbf{s}$ that it has sent to the base OT functionality, and use it to extract the inputs $\{(x_j^0, x_j^1)\}_{j=1}^m$, which are sent to the trusted party as the input of the corrupted sender. The execution then replaces the output of the honest receiver with the values $x_j^{r_j}$.

- **hyb_2:** This is like hyb_1, where here we change the definition of the base OTs functionality. It first receives $\mathbf{s}$ from the corrupted sender, sends it to the honest receiver, which then sends back to the trusted party uniform keys $\mathbf{k}_i^{s_i}$. The receiver also (locally) chooses keys $\mathbf{k}_i^{\overline{s_i}}$. Moreover, we compute the verification hashes $h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}}$ as $H(\mathbf{u}^i \oplus \mathbf{u}^j \oplus G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{s_\beta}))$ instead of $H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}}))$.

- **hyb_3:** This execution is like hyb_2, where for every pair $(\alpha,\beta) \in [\ell]^2$ we replace $h_{\alpha,\beta}^{s_\alpha,\overline{s_\beta}}, h_{\alpha,\beta}^{\overline{s_\alpha},s_\beta}$ with uniformly and independent random values of the appropriate length. Note that now each string $G(\mathbf{k}_i^{\overline{s_i}})$ appears only in the transmission of $\mathbf{u}^i$.

- **hyb_4:** Here, we change the input of the receiver when interacting with the adversary, and use the input 0^m instead of the string $\mathbf{r}$.

It is easy to see that hyb_0, hyb_1 and hyb_2 have the same output. The two executions hyb_2 and hyb_3 are indistinguishable since H is a κ-min entropy correlation robust function. In particular, the distinguisher receives the values $\{\mathbf{k}_i^{s_i}\}_{i=1}^{\ell}$, $(\mathbf{u}^1,\ldots,\mathbf{u}^\ell)$, as well as $\mathbf{r}$. Together with H, this uniquely determines the values $\{h_{\alpha,\beta}^{s_\alpha,s_\beta}, h_{\alpha,\beta}^{\overline{s_\alpha},\overline{s_\beta}}\}_{(\alpha,\beta)\in[\ell]^2}$. Moreover, from each $\mathbf{u}^i$ and $G(\mathbf{k}_i^{s_i})$, it can conclude $G(\mathbf{k}_i^{\overline{s_i}}) \oplus (0^m||\boldsymbol{\tau})$ as $\mathbf{u}^i \oplus G(\mathbf{k}_i^{s_i}) \oplus (\boldsymbol{\tau}||0^\kappa)$. In fact, for every $(\alpha,\beta) \in [\ell]^2$, it can conclude the value $\left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus (0^m||\boldsymbol{\tau})\right) \oplus G(\mathbf{k}_\beta^{s_\beta})$ and $G(\mathbf{k}_\alpha^{s_\alpha}) \oplus (G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus (0^m||\boldsymbol{\tau}))$. On the other hand, for every $(\alpha,\beta) \in [\ell]^2$ we have:

$$\begin{aligned} h_{\alpha,\beta}^{s_\alpha,\overline{s_\beta}} &= H(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus G(\mathbf{k}_\beta^{\overline{s_\beta}})) = H\left(t_{\alpha,\beta}^1 \oplus (0^m||\boldsymbol{\tau})\right) \\ h_{\alpha,\beta}^{\overline{s_\alpha},s_\beta} &= H(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus G(\mathbf{k}_\beta^{s_\beta})) = H\left(t_{\alpha,\beta}^2 \oplus (0^m||\boldsymbol{\tau})\right) \end{aligned}$$

where

$$t_{\alpha,\beta}^1 = \left(G(\mathbf{k}_\alpha^{s_\alpha}) \oplus (G(\mathbf{k}_\beta^{\overline{s_\beta}}) \oplus (0^m||\boldsymbol{\tau}))\right) \quad \text{and} \quad t_{\alpha,\beta}^2 = \left(G(\mathbf{k}_\alpha^{\overline{s_\alpha}}) \oplus (0^m||\boldsymbol{\tau})) \oplus G(\mathbf{k}_\beta^{s_\beta})\right) \ ,$$

and $t_{\alpha,\beta}^1$ and $t_{\alpha,\beta}^2$ are both known to the distinguisher. Since $\boldsymbol{\tau}$ is chosen uniformly from $\{0,1\}^\kappa$, assuming that H is a κ-min entropy strongly correlation robust function, the hash values $h_{\alpha,\beta}^{s_\alpha,\overline{s_\beta}}$ and $h_{\alpha,\beta}^{\overline{s_\alpha},s_\beta}$ are distributed uniformly in the respective domain.

As for hyb_3 and hyb_4, for every i the key $G(\mathbf{k}_i^{\overline{s_i}})$ is a one-time-pad for the string $\mathbf{r}$, and therefore hides it due to the pseudorandom property of G. ■