Actively Secure OT Extension with Optimal Overhead

Marcel Keller, Emmanuela Orsini, and Peter Scholl

Department of Computer Science, University of Bristol
mks.keller@gmail.com, emmanuela.orsini@esat.kuleuven.be,
peter.scholl@cs.au.dk

Abstract. We describe an actively secure OT extension protocol in the random oracle model with efficiency very close to the passively secure IKNP protocol of Ishai et al. (Crypto 2003). For computational security parameter κ, our protocol requires κ base OTs, and is the first practical, actively secure protocol to match the cost of the passive IKNP extension in this regard. The added communication cost is only additive in $O(\kappa)$, independent of the number of OTs being created, while the computation cost is essentially two finite field operations per extended OT. We present implementation results that show our protocol takes no more than 5% more time than the passively secure IKNP extension, in both LAN and WAN environments, and thus is essentially optimal with respect to the passive protocol.

Update, 2022: Roy (Crypto 2022) showed that Lemma 1, which the core of our proof relies on, is incorrect, so our protocol does not currently have a security proof. Roy also presented a protocol with an alternative consistency check and complete security proof, which also fixes issues with instantiating the hash function raised earlier by Guo et al. (IEEE S&P 2020) and Masny and Rindal (ACM CCS 2019). In Section 4, we show how to fix our protocol using the techniques by Roy.

1 Introduction

Oblivious transfer (OT) is a fundamental primitive in cryptography, used in the construction of a range of protocols. In particular, OT is sufficient and necessary for secure multi-party computation [28,11,16], and is also often used in special-purpose protocols for tasks such as private set intersection [24]. Due to a result of Impagliazzo and Rudich [13] it is very unlikely that OT is possible without the use of public-key cryptography, so all OT constructions have quite a high cost when used in a practical context.

This article is based on an earlier article: Actively Secure OT Extension with Optimal Overhead, published at *CRYPTO 2015* by Springer-Verlag, DOI: 10.1007/978-3-662-47989-6_35.

OT Extension. Since OT requires public key machinery, it is natural to wonder whether OT can be efficiently 'extended'. That is, starting with a small number of 'base OTs', create many more OTs with only symmetric primitives, somewhat analogous to the use of hybrid encryption to extend public key encryption. Beaver [3] first showed how, starting with κ base OTs, one could create $\mathsf{poly}(\kappa)$ additional OTs using only symmetric primitives, with computational security κ. Beaver's protocol is very elegant, but requires evaluation of pseudo-random generators within Yao's garbled circuits; therefore it is highly impractical. In 2003, Ishai, Kilian, Nissim and Petrank [14] presented a very efficient protocol for extending OTs, requiring only black-box use of symmetric primitives and κ base OTs. Concretely, the main cost of their basic protocol is computing and sending just *two* hash function values per OT. Asharov et al. [1] gave several algorithmic optimizations to the IKNP protocol, reducing the communication down to one hash value for the *random OT* variant, where the sender's messages are sampled at random, and using a cache-oblivious algorithm for matrix transposition, which turned out to be the computational bottleneck when implemented naively. By analyzing an implementation, they suggest that the actual bottleneck of the IKNP protocol is communication, particularly in wide area networks (WANs) with high latency and low bandwidth.

The above protocols are only secure against passive adversaries, who are trusted to strictly follow the protocol. For the case of actively secure protocols, which remain secure against arbitrary deviations from the protocol, typically most cryptographic protocols have a much greater cost than their passive counterparts. The actively secure OT extension of Ishai et al. [14] uses an expensive *cut-and-choose* technique, where s runs of the protocol are done in parallel to achieve $O(s)$ bits of statistical security. In recent years, the cost of actively secure OT extension has improved greatly, down to just constant overhead. The TinyOT protocol for secure two-party computation [22] is based on a very efficient OT extension where the total cost is roughly $\frac{8}{3}$ times the passively secure IKNP extension – this applies to communication and computation, as well as the number of base OTs required to run the protocol. Very recently, in an independent and concurrent work, Asharov et al. [2] gave a protocol reducing the overhead even further: their protocol requires roughly $\kappa + s$ base OTs for computational security κ and statistical security s, which in practice reduces the constant from $\frac{8}{3} \approx 2.7$ down to ≈ 1.4, plus an additive overhead in $O(\kappa)$.

Applications of OT extension. As we mentioned before, OT extension has been getting a lot of attention recently, because the efficiency of this procedure plays a decisive role in the overall efficiency of a number of protocols for secure computations where the number of OTs needed is very large, for example in the two-party and multiparty TinyOT protocols [22,19,5], in the MiniMAC protocol of Damgård et al. [8] and in private set intersection protocols [9,24].

Our Contributions. In this paper we give the first practical, actively secure OT extension protocol requiring only κ base OTs, matching the efficiency of the passively secure IKNP extension in this respect. For communication and computation costs, the overhead on top of IKNP is negligible: our protocol requires 2 finite field (of size κ) operations per extended OT, plus a small communication overhead of $O(\kappa)$ bits in a constant number of rounds, independent of the number of OTs being performed, which amortizes away when creating many OTs. We give extensive benchmarks (in both LAN and WAN settings) showing that the practical cost of our protocol for performing 10 million OTs is less than 6% more than the IKNP extension, and so is almost optimal. In contrast, the protocol of Asharov et al. [2] takes at least 80% more time than the passive protocol in the WAN setting and over 20% more in the LAN setting for 2^{23} OTs, according to their implementation figures.

The comparison table below shows the concrete efficiency of various other OT extension protocols, in terms of the number of base OTs required and the total communication and computation cost for creating ℓ OTs. Our protocol is more efficient than all previous protocols in all of these measures. Note that these comparisons are for OT extensions on strings of length at least κ bits. For shorter strings, the passively secure protocol of Kolesnikov and Kumaresan [17] is more efficient, but it does not seem straightforward to apply our techniques to obtain an actively secure protocol in that setting. We also omit the protocol of Ishai et al. [15], since although asymptotically this has only a constant overhead, the protocol is based on the 'MPC-in-the-head' technique, which has not been shown to be practical.

Protocol	Seed OTs	Comms.	Comp.	Security
[14]	128	$\ell \cdot 128$ bits	2ℓ hashes	passive, CRF
[14]	> 5000	$O(\ell \cdot \kappa \cdot s)$	$O(\ell \cdot s)$ hashing	active, CRF
[18]	323	$O(\ell \cdot \kappa^2)$	$O(\ell \cdot \kappa^2)$ XOR	active, CRF
[22]	342	$\ell \cdot 342$ bits + 43KB	$O(\ell)$ hashing	active, RO
[2]	170	$\ell \cdot 175$ bits + 22KB	$O(\ell)$ hashing	active, CRF
This work	128	$\ell \cdot 128$ bits + 10KB	$2\ell + 336$ hashes	active, RO

Table 1. Concrete cost of OT extension protocols for producing ℓ OTs of 128-bit strings with 128-bit computational and 40-bit statistical security parameters.

Our protocol is similar in structure to previous protocols [22,2], in that we carry out one run of the passively secure IKNP extension, and then use a *correlation check* to enforce correct behavior. As in the previous protocols, it is possible that a cheating receiver may pass our check, in which case some information on the sender's secret is leaked. However, the leakage is such that we still only need κ base OTs, and then must sacrifice $\kappa + s$ of the extended OTs produced from the IKNP extension, where s is a statistical security parameter, to ensure security. The check itself is extremely simple and only requires a constant number of hash computations on a fixed input length, unlike previous checks where the amount of data being hashed increases with the number of extended OTs.

Random Oracle Usage. We prove our OT extension protocol secure in the random oracle model, used for a functionality $\mathcal{F}_{\mathsf{Rand}}$, which securely generates random values, and the hash function H used to randomize the correlated OT outputs. For the function H, Ishai et al. [14] prove security of their protocol in the standard model, under the assumption that H is a *correlation robust function*. The protocol of Asharov et al. [2] is proven secure with the additional requirement that H satisfies some kind of leakage resilience, and it is conjectured that the protocol of Nielsen et al. [22] is also secure in this model.

Note that in the case of random OT, where the sender's outputs are defined as randomly chosen by the functionality, the security of the protocol (using the optimization of Asharov et al. [1], which cuts the communication in half) has only ever been proven in the random oracle model, because of the need for the simulator to program the receiver's outputs from H to be as defined by the functionality. Random OT can be used for an offline/online scenario where random OTs are generated in advance of the inputs being known, and is also often used in practical protocols (e.g. [24,22]), so we take the pragmatic approach of using random oracles for our security proofs, which also simplifies the exposition. However, due to the similarities between our protocol and previous ones [2,22], we believe it is likely that our (non-random) OT extension protocol can also be proven secure under a form of correlation robustness for H.

2 Preliminaries

2.1 Notation

We denote by κ the computational security parameter and by s the statistical security parameter. We let $\mathsf{negl}(\kappa)$ denote some unspecified function $f(\kappa)$, such that $f = o(\kappa^{-c})$ for every fixed constant c, saying that such a function is *negligible* in κ. We say that a probability is *overwhelming* in κ if it is $1-\mathsf{negl}(\kappa)$. We denote by $a \overset{\$}{\leftarrow} A$ the random sampling of a from a distribution A, and by $[d]$ the set of elements $\{1, \dots d\}$.

Throughout the proofs we will often identify $\mathbb{F}_2^\kappa$ with the finite field $\mathbb{F}_{2^\kappa}$. Addition is the same in both; we will use " $\cdot$ " for multiplication in $\mathbb{F}_{2^\kappa}$ and " $*$ " for the component-wise product in $\mathbb{F}_2^\kappa$. We use lower case letters to denote elements in $\mathbb{F}_2$ and bold lower case letters for vectors in $\mathbb{F}_2^\kappa$ and elements in $\mathbb{F}_{2^\kappa}$. We will use the notation $\mathbf{v}[i]$ to denote the i-th entry of $\mathbf{v}$.
Given a matrix A, we denote its rows by subindices $\mathbf{a}_i$ and its columns by superindices $\mathbf{a}^k$. Given a vector $\mathbf{v} \in \mathbb{F}_2^\kappa$, we denote by $\bar{\mathbf{v}}$ the vector in $\mathbb{F}_2^\kappa$ such that $\mathbf{v} + \bar{\mathbf{v}} = \mathbf{1}$. We say that a vector $\mathbf{v} \in \mathbb{F}_2^\kappa$ is *monochrome* if $v[i] = v[j]$, for each $i, j \in [\kappa]$; otherwise we say it is *polychrome*.

In our proofs we often use the notion of affine space. We recall that an affine space is a set X that admits a free transitive action of a vector space V.

2.2 Oblivious Transfer and OT extension

Oblivious transfer (OT) [27,25,10,4] is a two-party protocol between a *sender* S and a *receiver* R. The sender transmits part of its input to R, in such a way that S remains oblivious as what part of its input was transmitted and R does not obtain more information than it is entitled.

We use three main oblivious transfer functionalities. We denote by $\mathcal{F}_{\mathsf{OT}}$ the standard $\binom{2}{1}$-OT functionality, where the sender S inputs two messages $\mathbf{v}_0, \mathbf{v}_1 \in \mathbb{F}_2^\kappa$, and the receiver inputs a choice bit x, and at the end of the protocol R learns only the selected message $\mathbf{v}_x$. We use the notation $\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$ to denote the functionality that provides ℓ $\binom{2}{1}$-OTs of messages in $\mathbb{F}_2^\kappa$ (see Fig. 1 for a formal definition). Another variant of OT is correlated OT, where the sender's messages are correlated, i.e. $\mathbf{v}_0 + \mathbf{v}_1 = \Delta$ for a fixed $\Delta \in \mathbb{F}_2^\kappa$; in Fig. 3 we give a version of this functionality which allows "errors". Finally, in the random OT functionality , $\mathcal{F}_{\mathsf{ROT}}$, the messages $\mathbf{v}_0, \mathbf{v}_1$ are sampled uniformly at random by the functionality (Fig. 7).

Functionality $\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$

$\mathcal{F}$ running with R, S and an adversary $\mathcal{S}$ proceeds as follows:

- The functionality waits for input $(\mathbf{v}_{0,i}, \mathbf{v}_{1,i}) \in \mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa$, $i \in [\ell]$, from S and $x_1, \ldots, x_\ell$, with $x_i \in \mathbb{F}_2$, from R.
- It outputs $\mathbf{v}_{x_i,i}$, $i \in [\ell]$, to R.

Fig. 1. The OT functionality

Functionality $\mathcal{F}_{\mathsf{COTe}}^{\kappa,\ell}$

The functionality is parametrized by the number ℓ of resulting OTs and by the key length κ.
Running with parties S, R, and an adversary $\mathcal{A}$ it operates as follows.

Initialize: Upon receiving Δ from S, where $\Delta \in \mathbb{F}_{2^\kappa}$, the functionality stores Δ.
Extend:

- Upon receiving $(\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$ from R, where $\mathbf{x}_i \in \mathbb{F}_{2^\kappa}$, sample $\mathbf{t}_j \in \mathbb{F}_{2^\kappa}$, $j = 1, \ldots, \ell$, and output them to R. Compute $\mathbf{q}_j = \mathbf{t}_j + \mathbf{x}_j * \Delta$, $j = 1, \ldots, \ell$, and output them to S.
- If R is corrupt, wait for $\mathcal{A}$ to input $\mathbf{t}_j$ and output as before.

Fig. 2. Correlated OT with errors functionality $\mathcal{F}_{\mathsf{COTe}}$

IKNP Protocol Augmented with Errors. In Fig. 2, we model the IKNP extension as a separate functionality, $\mathcal{F}_{\mathsf{COTe}}$ that incorporates a cheating receiver's behavior, and call this *correlated OT with errors*. Fig. 3 gives the implementation of this functionality: after the first phase and the local expansion of the seeds through a pseudorandom generator PRG, R holds two $\ell \times \kappa$ matrices $\{\mathbf{t}_0^i\}_{i\in[\kappa]}, \{\mathbf{t}_1^i\}_{i\in[\kappa]}$, while S holds the vector $\Delta \in \mathbb{F}_2^\kappa$ and the matrix $\{\mathbf{t}_{\Delta_i}^i\}_{i\in[\kappa]}$. In the extension phase, we allow a cheating receiver R to input vectors $\mathbf{x}_1, \dots, \mathbf{x}_\ell \in \mathbb{F}_2^\kappa$, instead of inputting bits $x_1, \dots, x_\ell$. To better understand this situation we can imagine R inputting an $\ell \times \kappa$ matrix X, having $\mathbf{x}_1, \dots, \mathbf{x}_\ell \in \mathbb{F}_2^\kappa$ as rows and $\mathbf{x}^1, \dots, \mathbf{x}^\kappa \in \mathbb{F}_2^\ell$ as columns. If R is honest then $\mathbf{x}^1 = \dots = \mathbf{x}^\kappa$ and the rows $\mathbf{x}_j$ are "monochrome" vectors, i.e. consisting either of all 0's or all 1's. At this point the receiver computes $\mathbf{u}^i = \mathbf{t}_0^i + \mathbf{t}_1^i + \mathbf{x}^i$, for each $i \in [\kappa]$. Clearly, if R is honest, they send the same vector $\mathbf{x}^i$ for each i. After this step S computes $\mathbf{q}^i = \mathbf{t}_{\Delta_i}^i + \mathbf{u}^i + \mathbf{u}^i \cdot \Delta_i = \mathbf{t}_0^i + \mathbf{x}^i \cdot \Delta_i$, obtaining the $\ell \times \kappa$ matrix Q, having $\mathbf{q}^i$ as columns and $\mathbf{q}_j = \mathbf{t}_{0,j} + \mathbf{x}_j * \Delta$ as rows. If $\mathbf{x}_j$ is monochrome, i.e $\mathbf{x}_j = x_j \cdot (1, \dots, 1)$, then $\mathbf{q}_j = \mathbf{t}_{0,j} + x_j \cdot \Delta$, otherwise, rewriting $\mathbf{x}_j$ as $\mathbf{x}_j = x_j \cdot (1, \dots, 1) + \mathbf{e}_j$, we get $\mathbf{q}_j = \mathbf{t}_{0,j} + x_j \cdot \Delta + \mathbf{e}_j * \Delta$, where $\mathbf{e}_j$ is an "error" vector counting the number of positions in which R cheated.

Notice that, compared with the original IKNP protocol, the protocol COTe stops before hashing the output with the random oracle to break the correlation and performing the final round of communication. It is easy to see (and was shown e.g. by Nielsen [21]) that the protocol for COTe (given in Fig. 3) securely implements this functionality.

3 Our Actively Secure OT Extension Protocol

In this section we describe our protocol for actively secure OT extension based on the passive IKNP functionality, $\mathcal{F}_{\mathsf{COTe}}$. We recall that to deal with malicious adversaries, all the known actively secure OT extension protocols add a consistency check to the passive secure IKNP protocol to ensure that R inputs consistent values.

For example, in previous works [22,2] this check is added before the "extension" phase, i.e. before the sender S "reverses" the base OTs and breaks the correlation, effectively checking on the OT seeds. In our construction we check the correlation for consistency *after* the extension step, precisely after the execution of COTe, actually checking the extended OTs.

The high level idea of our protocol in Fig. 7 is to perform a simple correlation check to ensure that the receiver used the same vector $\mathbf{x}^i$ for each $\mathbf{u}^i$ sent in Step 3 of the IKNP extension. If the check passes, then the correlated OTs are hashed to obtain random OTs. This check requires sacrificing $\kappa + s$ extended OTs to ensure security, so we obtain a reduction from $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$ to $\mathcal{F}_{\mathsf{COTe}}^{\kappa,\ell'}$, with $\ell' = \ell + (\kappa + s)$.

The intuition in this reduction is that, if the check passes, the adversary can only learn few bits of the correlation vector Δ, and hence the values $H(\mathbf{q}_j + \Delta)$ are actually random except with negligible probability. Finally, if required, the

Protocol for $\mathsf{COTe}^{\kappa,\ell}$

Initialize: This is independent of inputs and only needs to be done once.

1. R samples κ pairs of κ-bit seeds, $\{(\mathbf{k}_0^i, \mathbf{k}_1^i)\}_{i=1}^{\kappa}$.
2. S samples a random κ-bit string Δ.
3. The parties call $\kappa \times \mathsf{OT}_\kappa$ with inputs Δ and $\mathbf{k}_0, \mathbf{k}_1$.
4. S receives $\mathbf{k}_{\Delta_i}^i$ for $i = 1, \ldots, \kappa$.

Extend: This creates ℓ extended C-OTs. Note that this phase can be iterated, as done by Asharov et al. [1].

1. R inputs monochrome vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\ell$. Let $x_1, \ldots, x_\ell$ be the bits of the vectors for the case when R is honest.
2. Expand $\mathbf{k}_i^0$ and $\mathbf{k}_i^1$ using a pseudo random generator (PRG), letting
$$\mathbf{t}_0^i = \mathsf{PRG}(\mathbf{k}_i^0) \in \mathbb{F}_2^\ell \quad \text{and} \quad \mathbf{t}_1^i = \mathsf{PRG}(\mathbf{k}_i^1) \in \mathbb{F}_2^\ell, \quad i = 1, \ldots, \kappa.$$
so R knows $(\mathbf{t}_i^0, \mathbf{t}_i^1)$ and S knows $\mathbf{t}_{\Delta_i}^i$ for $i = 1, \ldots, \kappa$.
3. R computes
$$\mathbf{u}^i = \mathbf{t}_0^i + \mathbf{t}_1^i + \mathbf{x}^i \in \mathbb{F}_2^\ell, \quad i = 1, \ldots, \kappa,$$
where $\mathbf{x}^i = (x_1, \ldots, x_\ell) \in \mathbb{F}_2^\ell$ and sends them to S. Here we are creating the keys correlation that permits to extend OTs, inverting the role of sender and receiver.
4. S computes
$$\mathbf{q}^i = \Delta_i \cdot \mathbf{u}^i + \mathbf{t}_{\Delta_i}^i \in \mathbb{F}_2^\ell.$$
Notice that $\mathbf{q}^i = \mathbf{t}_0^i + \Delta_i \cdot \mathbf{x}^i$, for $i = 1, \ldots, \kappa$.
5. Let $\mathbf{q}_j$ denote the j-th row of the $\ell \times \kappa$ bit matrix $Q = [\mathbf{q}^1 | \ldots | \mathbf{q}^\kappa]$, and similarly let $\mathbf{t}_j$ be the j-th row of $[\mathbf{t}_0^1 | \ldots | \mathbf{t}_0^\kappa]$. Note that
$$\mathbf{q}_j = \mathbf{t}_j + \mathbf{x}_j * \Delta, \quad j = 1, \ldots, \ell.$$

Output: R outputs $\mathbf{t}_j$, S outputs $\mathbf{q}_j$ and Δ.

Fig. 3. Protocol for correlated OT with errors between S and R.

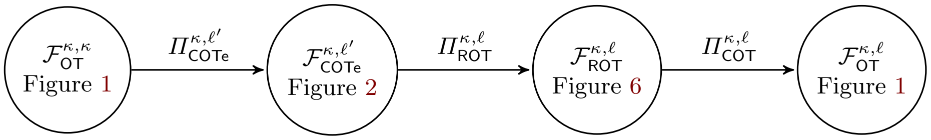
$\mathcal{F}_{\mathsf{OT}}^{\kappa,\kappa}$
Figure 1
$\Pi_{\mathsf{COTe}}^{\kappa,\ell'}$
$\mathcal{F}_{\mathsf{COTe}}^{\kappa,\ell'}$
Figure 2
$\Pi_{\mathsf{ROT}}^{\kappa,\ell}$
$\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$
Figure 6
$\Pi_{\mathsf{COT}}^{\kappa,\ell}$
$\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$
Figure 1

Fig. 4. Relationship between the different functionalities used to go from $\mathcal{F}_{\mathsf{OT}}^{\kappa,\kappa}$ to $\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$.

random OTs obtained from ROT can be derandomized with an additional set of messages from the sender, using the standard reduction from $\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$ to $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$.

The relationship between all the functionalities used are described in Fig. 4. The first stage to $\mathcal{F}_{\mathsf{COTe}}$ essentially consists of the IKNP OT extension protocol (with some modifications from the protocol by Asharov et al. [1]) that we have seen in the previous section.

Functionality $\mathcal{F}_{\mathsf{Rand}}^{\mathbb{F}}$

Random sample: Upon receiving $(rand; u)$ from all parties, it samples a uniform $r \in \mathbb{F}$ and outputs $(rand, r)$ to all parties.

Fig. 5. Functionality $\mathcal{F}_{\mathsf{Rand}}^{\mathbb{F}}$

3.1 Protocol from COTe to ROT

Here we describe the protocol implementing the $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$ functionality in Fig. 6. The main idea of our construction is to use a variant of the MAC check protocol from SPDZ [7], adapted for two parties where one party holds the MAC key, to check the correlation is consistent. The correlation check is performed on the ℓ' correlated OTs of length κ output by $\mathcal{F}_{\mathsf{COTe}}^{\kappa,\ell'}$, i.e. after the vectors have been transposed. Recall that after running $\mathcal{F}_{\mathsf{COTe}}$, the sender S has $\Delta, \mathbf{q}_1, \ldots, \mathbf{q}_{\ell'} \in \mathbb{F}_2^\kappa$ and the receiver R has $\mathbf{x}_1, \ldots, \mathbf{x}_{\ell'}, \mathbf{t}_1, \ldots, \mathbf{t}_{\ell'} \in \mathbb{F}_2^\kappa$ such that $\mathbf{q}_j = \mathbf{t}_j + \mathbf{x}_j * \Delta$ for $j \in [\ell']$. If R was honest then every $\mathbf{x}_j$ is monochrome, so $\mathbf{q}_j = \mathbf{t}_j + x_j \cdot \Delta$ for bits $x_1, \ldots, x_{\ell'}$.

To carry out the check, both parties first securely generate ℓ' random weights $\chi_1, \ldots, \chi_{\ell'} \in \mathbb{F}_2^\kappa$, and then compute weighted sums of their outputs from $\mathcal{F}_{\mathsf{COTe}}$. Then R sends these values to S to check consistency with S's output. So, R computes $x = \sum_{j=1}^{\ell'} x_j \cdot \chi_j, t = \sum_{j=1}^{\ell'} \mathbf{t}_j \cdot \chi_j$ and S computes $q = \sum_{j=1}^{\ell'} \mathbf{q}_j \cdot \chi_j$, where the vectors $\mathbf{t}_j, \mathbf{q}_j, \chi_j$ are viewed as elements of $\mathbb{F}_{2^\kappa}$ and multiplications are performed in this finite field. S then checks that $q = t + x \cdot \Delta$.

Clearly, by linearity of the correlated OT output, the check will always pass for an honest receiver. If R is corrupted then it is possible they may pass the check despite having used polychromatic $\mathbf{x}_j$ vectors; in this case they will learn

Functionality $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$

The functionality is parametrized by the number ℓ of resulting OTs and by the length of the OT strings κ.
Running with parties S, R and an ideal adversary denoted by $\mathcal{S}$, it operates as follows.

- Upon receiving $(R, (x_1, \ldots, x_\ell))$ from R, where $x_j \in \mathbb{F}_2$, the functionality samples random $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j}) \in \mathbb{F}_{2^\kappa}^2$, for $j \in [\ell]$. Then it sends $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j})$ to S and $\mathbf{v}_{x_j,j}$ to R.
- If R is corrupt: if $\mathcal{S}$ inputs Abort, $\mathcal{F}$ sends Abort to S and it halts. Otherwise it waits for $\mathcal{S}$ to input x_j for all $j \in [\ell]$. Then it samples random $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j})$, $j \in [\ell]$ and outputs them to S. It also sends $\mathbf{v}_{x_j,j}$ to $\mathcal{S}$ for all $j \in [\ell]$.
- If S is corrupt it waits for $\mathcal{S}$ to input $(\mathbf{v}_{0,j}, \mathbf{v}_{1,j}), j \in [\ell]$, and then outputs as above using these values.

Fig. 6. Functionality $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$

some information about Δ. We show that this leakage is *optimal*, in the sense that a cheating receiver can learn c bits of information on Δ with at most probability 2^{-c}, and the possible errors in the resulting OTs do not provide the adversary with any further useful information. Looking ahead to the proof, the success probability of a receiver who passes the check in breaking the resulting OTs with $q = \mathsf{poly}(\kappa)$ queries to H will therefore be $q/2^{\kappa-c}$, giving an overall success probability of $q/2^\kappa$. This implies that κ base OTs suffice for computational security κ.

On the other hand, if the sender is corrupted, our correlation check introduces the possibility that the values of x and t could leak information about R's input bits $x_1, \ldots, x_\ell$. However, we show that it suffices to perform $\kappa + s$ additional OTs with random choice bits to counter against this leakage, for statistical security s. Overall, this means our protocol requires only κ base OTs, which is optimal with respect to the IKNP extension, and an additive overhead of $s + \kappa$ extended OTs, regardless of the number ℓ of OTs required, as well as just $O(\kappa)$ additional communication in a constant number of rounds.

3.2 Analysis of the Correlation Check

Corrupt Sender. To ensure that the correlation check step is secure against a corrupt sender we must carefully choose the parameter ℓ', which determines the size of the batch each check is performed on. Recall that the elements in the field $\mathbb{F}$ are κ bits long; if $\ell' \leq \kappa$ then it is likely that the secret bits x_j will be uniquely determined given χ_j and x, so an adversary could attempt to solve the corresponding knapsack problem to recover these. As we will see in the proof in Theorem 1, to thwart this attack, we use a technical lemma giving a bound on the rank of a random binary matrix. This is also the reason why we do not let the sender sample $\{\chi_j\}_{j=1}^{\ell}$.

Corrupt Receiver. The case of a corrupt receiver is much more involved. We now investigate a cheating receiver's success probability in the correlation check stage of the ROT protocol in Fig. 7. Let $\mathbf{x}_1, \ldots, \mathbf{x}_{\ell'}$ be the vectors in $\mathbb{F}_2^\kappa$ input by R during the protocol. Taking these to be the rows of a $\ell' \times \kappa$ matrix, let $\mathbf{x}^1, \ldots, \mathbf{x}^\kappa$ be the *columns* of the same matrix, in $\mathbb{F}_2^{\ell'}$. If R was honest then $\{\mathbf{x}_j\}_{j\in[\ell']}$ are all monochrome and $\{\mathbf{x}^i\}_{i\in[\kappa]}$ are all equal. The following Lemma gives the main properties needed from our correlation check.

Protocol for $\mathsf{ROT}^{\kappa,\ell}$

Let $\ell' = \ell + (\kappa + s)$. $\mathcal{F}_{\mathsf{COTe}}^{\kappa,\ell'}$, henceforth denoted as $\mathcal{F}_{\mathsf{COTe}}$.

Initialize: The parties call $\mathcal{F}_{\mathsf{COTe}}.\mathsf{Initialize}$ where S inputs $\Delta \in \mathbb{F}_2^\kappa$.

Extend: The parties call $\mathcal{F}_{\mathsf{COTe}}.\mathsf{Extend}$, where R inputs monochrome vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{\ell'} \in \mathbb{F}_2^\kappa$ such that $\mathbf{x}_j = x_j \cdot (1, \ldots, 1)$ for $j \in [\ell]$ and $\mathbf{x}_j = x_j \cdot (1, \ldots, 1)$ for random $x_j \in \mathbb{F}_2$ for $j \in [\ell + 1, \ell']$.

S receives $\mathbf{q}_j \in \mathbb{F}_2^\kappa$ and R receives $\mathbf{t}_j \in \mathbb{F}_2^\kappa$ for $j = 1, \ldots, \ell'$ such that:

$$\mathbf{t}_j = \mathbf{q}_j + \mathbf{x}_j * \Delta$$

Check correlation: We check that the vectors input by R during the C-OT were monochrome, by checking a random linear combination. If R cheated earlier she could learn a few bits of Δ here, causing some leakage.

- Sample $(\chi_1, \ldots, \chi_{\ell'}) \leftarrow \mathcal{F}_{\mathsf{Rand}}(\mathbb{F}_{2^\kappa}^{\ell'})$.
- R computes

$$x = \sum_{j=1}^{\ell'} x_j \cdot \chi_j \quad \text{and} \quad t = \sum_{j=1}^{\ell'} \mathbf{t}_j \cdot \chi_j$$

and sends these to S.

- S computes

$$q = \sum_{j=1}^{\ell'} \mathbf{q}_j \cdot \chi_j$$

and checks that $t = q + x \cdot \Delta$. If the check fails, output Abort, otherwise S outputs $\Delta, \{\mathbf{q}_j\}_{j\in[\ell]}$ and R outputs $\{\mathbf{t}_j, x_j\}_{j\in[\ell]}$.

Randomize: Now break the correlation and remove any leaking bits of Δ.

- R sets

$$\mathbf{v}_{x_j,j} = H(j \| \mathbf{t}_j),$$

and outputs $x_j, \mathbf{v}_{x_j,j}$, $j \in [\ell]$.

- S outputs

$$\mathbf{v}_{0,j} = H(j \| \mathbf{q}_j) \quad \text{and} \quad \mathbf{v}_{1,j} = H(j \| \mathbf{q}_j + \Delta), \quad j \in [\ell].$$

Fig. 7. Random OT extension protocol from correlated OT with errors.

Lemma 1. *Let $S_\Delta \subseteq \mathbb{F}_2^\kappa$ be the set of all Δ for which the correlation check passes, given the view of the receiver. Except with probability $2^{-\kappa}$, there exists $k \in \mathbb{N}$ such that*

1. $|S_\Delta| = 2^k$.
2. *For every $\mathbf{s} \in \{\mathbf{x}^i\}_{i\in[\kappa]}$, let $H_\mathbf{s} = \{i \in [\kappa] \mid \mathbf{s} = \mathbf{x}^i\}$. Then one of the following holds:*
   - *For all $i \in H_\mathbf{s}$ and any $\Delta^{(1)}, \Delta^{(2)} \in S_\Delta$, $\Delta_i^{(1)} = \Delta_i^{(2)}$.*
   - *$k \leq |H_\mathbf{s}|$, and $|\{\Delta_{H_\mathbf{s}}\}_{\Delta\in S_\Delta}| = 2^k$, where $\Delta_{H_\mathbf{s}}$ denotes the vector consisting of the bits $\{\Delta_i\}_{i\in H_\mathbf{s}}$. In other words, S_Δ restricted to the bits corresponding to $H_\mathbf{s}$ has entropy at least k.*

   Furthermore, there exists $\hat{\mathbf{s}}$ such that $k \leq |H_{\hat{\mathbf{s}}}|$.

Proof. Roy [26] proved the lemma to be false.

We now give some intuition about the meaning of this statement. The set S_Δ is the set of all possible values of Δ with which the correlation check could pass – note that since Δ is uniformly random to the receiver, their probability of passing the check is therefore $|S_\Delta|/2^\kappa$. For some vector $\mathbf{s} \in \{\mathbf{x}^i\}_{i\in[\kappa]}$, the set $H_\mathbf{s}$ represents indices of all of the vectors equal to $\mathbf{s}$. Clearly, for an honest receiver, $H_\mathbf{s}$ is always just the set $\{1, \ldots, \kappa\}$, and so the size of $H_\mathbf{s}$ measures the amount of deviation in the protocol for a given $\mathbf{s}$. The precise indices in $H_\mathbf{s}$ are also important, as they correspond to a subset of the bits of the secret Δ, which could be learnt using $2^{|H_\mathbf{s}|}$ queries to the hash function (causing the simulation to abort in our security proof).

The second part of the lemma implies that *for any* $\mathbf{s}$, either the bits of Δ corresponding to the indices in $H_\mathbf{s}$ are constant for all possible $\Delta \in S_\Delta$, or, the size of $H_\mathbf{s}$ is at least k, which means the corresponding abort in the simulation occurs with probability at least $1-2^{-k+\kappa}$. Clearly in the first case, the adversary gains no new information, but in the second case we have a bound on the amount of information an adversary can learn, which directly corresponds to the size of the set S_Δ, and hence also the success probability in the correlation check. The final part of the Lemma, concerning $\hat{\mathbf{s}}$, simply states that there is always a vector $\mathbf{s}$ that satisfies the second condition, so at least one block of k bits of Δ remains hidden. A careful analysis of these possible deviations allows us to show that κ base OTs suffice for our protocol.

3.3 Proof of Security

Theorem 1. *The protocol in Fig. 7 securely implements the $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$ functionality in the $(\mathcal{F}_{\mathsf{COTe}}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{RO}})$-hybrid model with computational security parameter κ.*

The computational security parameter κ manifests itself in that the adversary is only allowed $\mathsf{poly}(\kappa)$ calls of the random oracle in the proof. Other than that, the simulation is statistically indistinguishable.

Simulator for $\mathsf{ROT}^{\ell,\kappa}$

Initialize: If S is corrupt, receive $\varDelta \in \mathbb{F}_{2^\kappa}$ from $\mathcal{A}$ and store $\varDelta$.

OT extension: For the sake of simplicity we describe separately the case when R is corrupt and the case when S is corrupt.

Corrupt R:

1. Receive vectors $\mathbf{x}_1, \dots, \mathbf{x}_{\ell'}$ and $\mathbf{t}_1, \dots, \mathbf{t}_{\ell'} \in \mathbb{F}_{2^\kappa}$ from $\mathcal{A}$.
2. Emulating $\mathcal{F}_{\mathsf{COTe}}$ with a dummy sender, sample a random $\varDelta \in \mathbb{F}_{2^\kappa}$.
3. Sample random $\chi_1, \dots, \chi_{\ell'} \in \mathbb{F}_{2^\kappa}$ and send these values to $\mathcal{A}$.
4. $\mathcal{S}$ receives x, t from $\mathcal{A}$ and carries out the correlation check. If the check fails, the $\mathcal{S}$ sends Abort to $\mathcal{F}_{\mathsf{ROT}}$ and it halts.
5. Let $\hat{\mathbf{s}}$ and $H_{\hat{\mathbf{s}}}$ be as in Lemma 1, that is, $k \leq |H_{\hat{\mathbf{s}}}|$, and $\mathbf{x}^i = \mathbf{x}^{i'}$ for all $i, i' \in H_{\hat{\mathbf{s}}}$. This implies that $\mathbf{x}_j[i] = \mathbf{x}_j[i']$ for all $i, i' \in H_{\hat{\mathbf{s}}}$ and $j \in [\ell]$. For each $j \in [\ell]$, set $x_j = \mathbf{x}_j[i]$ for some $i \in H_{\hat{\mathbf{s}}}$.
6. Send x_j to $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$ and receive $\mathbf{v}_{\mathbf{x}_j,j}$ for all $j \in [\ell]$.
7. Emulate the random oracle queries as follows:
   - If the input is $(j \| \mathbf{t}_j + \mathbf{x}_j * \varDelta + x_j \cdot \varDelta)$, return $\mathbf{v}_{x_j,j}$.
   - If the input is $(j \| \mathbf{t}_j + \mathbf{x}_j * \varDelta + \overline{x_j} \cdot \varDelta)$, abort.
   - Otherwise, return a random value from $\mathbb{F}_2^\kappa$ (consistent with previous queries).

Corrupt S:

1. Receive $\mathbf{q}_1, \dots, \mathbf{q}_{\ell'}$ from $\mathcal{A}$.
2. Sample random $\chi_1, \dots, \chi_{\ell'} \in \mathbb{F}_{2^\kappa}$ and send these to $\mathcal{A}$.
3. Compute $q = \sum_{j=1}^{\ell'} \mathbf{q}_j \cdot \chi_j$, sample random $x \in \mathbb{F}_{2^\kappa}$ and compute $t = q + x \cdot \varDelta$.
4. Output whatever $\mathcal{A}$ outputs and halt.

Fig. 8. Simulator for random OT extension

Proof. We construct a simulator $\mathcal{S}$ that has access to $\mathcal{F}_{\mathsf{ROT}}$, and show that no environment $\mathcal{Z}$ can distinguish between an interaction with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{ROT}}$ and an interaction with the real adversary $\mathcal{A}$ and the real parties. To simulate a real world execution of the protocol, $\mathcal{S}$ starts an internal copy of $\mathcal{A}$ and runs an internal copy of the protocol with dummy parties π_S and π_R, as shown in Figure 8.

First we deal with the (simpler) case of a corrupt sender. Since the simulator gets the sender's secret Δ, it is straightforward to construct x and t that will pass the check. All we need to do is argue indistinguishability from the real world execution. We need the following lemma.

Lemma 2. *Let A be a random $(\kappa + m) \times \kappa$ matrix over $\mathbb{F}_2$, where $m > 0$. Then A has rank κ except with probability less than 2^{-m}.*

Proof. See Section A.

Recall that in the real world the sender receives

$$x = \sum_{j=1}^{\ell'} x_j \cdot \chi_j = \sum_{j=1}^{\ell} x_j \cdot \chi_j + \sum_{j=\ell+1}^{\ell'} x_j \cdot \chi_j .$$

The second summation corresponds to the image of a linear map from $\mathbb{F}_2^{\ell'-\ell} = \mathbb{F}_2^{\kappa+s}$ to $\mathbb{F}_2^{\kappa}$. From Lemma 2, it follows that this map has full rank with probability $1 - 2^{-s}$. In this case, the second summation is uniformly random in $\mathbb{F}_{2^\kappa}$ because $(x_{\ell+1}, \ldots, x_{\ell'})$ were chosen uniformly at random by R, and so indistinguishable from the simulated random x. Finally, t has the same distribution in both worlds because there is only one t fulfilling the equation $q = t + x \cdot \Delta$.

We now consider the case of R being corrupted. In steps 1-4, $\mathcal{S}$ simply emulates $\mathcal{F}_{\mathsf{COTe}}$ and the correlation check, choosing random values for the dummy sender's input. Lemma 1 states that (except with negligible probability) $|S_\Delta| = 2^k$ for some $k \in \mathbb{N}$. For every $\mathbf{s} \in \{\mathbf{x}^i\}_{i \in [\kappa]}$, let $H_{\mathbf{s}} = \{i \in [\kappa] \mid \mathbf{s} = \mathbf{x}^i\}$ and $\hat{\mathbf{s}}$ as in Lemma 1. Recall that the adversary knows $(\mathbf{t}_j, \mathbf{x}_j)$ such that $\mathbf{t}_j = \mathbf{q}_j + \mathbf{x}_j * \Delta$. If $x_1, \ldots, x_\ell$ are the bits of $\hat{\mathbf{s}}$ then this can be expressed as $\mathbf{t}_j = \mathbf{q}_j + x_j \cdot \Delta + \mathbf{e}_j * \Delta$, where $\mathbf{e}_j = (x_j, \ldots, x_j) + \mathbf{x}_j$ is an adversarially chosen error vector. By definition, $\mathbf{e}_j[i] = \mathbf{e}_j[i']$ for all $i, i' \in H_{\mathbf{s}}$, for any $\mathbf{s} \in \{\mathbf{x}^i\}_{i \in [\kappa]}$ and $j \in [\ell]$.

In step 7, the simulator responds to the adversary's random oracle queries. Notice that it is the queries $\mathbf{q}_j = \mathbf{t}_j + \mathbf{x}_j * \Delta$ and $\mathbf{q}_j + \Delta = \mathbf{t}_j + \overline{\mathbf{x}_j} * \Delta$ that require the reply conforming to the output of $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$. The simulator knows $\mathbf{v}_{x_j,j}$, which is the output of $H(j \| \mathbf{q}_j + x_j \cdot \Delta)$ in the real-world protocol. On the other hand, if the adversary queries $(j \| \mathbf{q}_j + \overline{x_j} \cdot \Delta)$, the simulator cannot give the right output and thus aborts.

We now investigate the probability that this happens, given that the correlation check has passed. It holds that

$$\mathbf{q}_j + \overline{x_j} \cdot \Delta = \mathbf{t}_j + \mathbf{x}_j * \Delta + \overline{x_j} \cdot \Delta = \mathbf{t}_j + (\mathbf{x}_j + (\overline{x_j}, \ldots, \overline{x_j})) * \Delta .$$

For $i \in H_{\hat{\mathbf{s}}}$, $\mathbf{x}_j[i] = x_j$ and thus $(\mathbf{x}_j + (\overline{x_j}, \ldots, \overline{x_j})[i] = 1$. By Lemma 1, there are $|S_\Delta| = 2^k$ possibilities for $(\mathbf{x}_j + (\overline{x_j}, \ldots, \overline{x_j})) * \Delta$ and hence $\mathbf{q}_j + \overline{x_j} \cdot \Delta$, given $\Delta \in S_\Delta$. Therefore, the probability of one such query is 2^{-k}.

However, we must also show that the environment cannot learn any additional information from previous queries. For example, when R queries $\mathbf{q}_j + x_j \cdot \Delta$ to get their correct OT output, the environment (who sees the honest sender output so can verify this has occurred) can learn $\mathbf{e}_j * \Delta$ by computing $\mathbf{q}_j + x_j \cdot \Delta + \mathbf{t}_j$. By definition, the bits of $\mathbf{e}_j$ corresponding to any index set $H_\mathbf{s}$ are constant. Furthermore, Lemma 1 states that either $\Delta_i^{(1)} = \Delta_i^{(2)}$ for all $\Delta^{(1)}, \Delta^{(2)} \in S_\Delta$ and $i \in H_\mathbf{s}$ or $|H_\mathbf{s}| \geq k$. In the first case, $e_j[i] \cdot \Delta_i$ is known by the fact that $\Delta \in S_\Delta$. In the second case, consider that $e_j[i]$ is the same for all $i \in H_\mathbf{s}$. If $e_j[i] = 0$ for all $i \in H_\mathbf{s}$, then $e_j[i] \cdot \Delta_i = 0$ for all $i \in H_\mathbf{s}$. On the other hand, if $e_j[i] = 1$, there are 2^k possibilities for $\mathbf{e}_j * \Delta$ (given $\Delta \in S_\Delta$ as above) and thus for $\mathbf{q}_j + x_j \cdot \Delta$. Hence, either the latter is known to the adversary already or the probability of querying it is 2^{-k} per query.

It follows that the probability the simulation aborts after the correlation check has passed is at most $q \cdot 2^{-k}$, where q is the number of queries made by the environment. Now taking into account the fact that the check passes with probability $|S_\Delta| \cdot 2^{-\kappa} + 2^{-\kappa} = 2^{-\kappa} \cdot (2^k + 1)$, the overall success probability of distinguishing is at most $q \cdot 2^{-\kappa} \cdot (1 + 2^{-k})$, which is negligible in κ.

□

3.4 From ROT to OT

Finally we show how to reduce $\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$ to $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$.

Lemma 3. *The protocol in Fig. 9 securely implements the $\mathcal{F}_{\mathsf{OT}}^{\kappa,\ell}$ functionality in the $\mathcal{F}_{\mathsf{ROT}}^{\kappa,\ell}$-hybrid model.*

Proof. It is easy to describe a simulator for a corrupt R. $\mathcal{S}$ runs a copy of $\mathcal{A}$ setting dummy parties π_R and π_S and then simulates for them a real execution of DeROT, running an internal copy of $\mathcal{F}_{\mathsf{ROT}}$.

We just need to show indistinguishably of the transcripts and of the outputs. In both worlds, $\{\mathbf{d}_{x_i,i}\}_{i \in [\ell]}$ and $\{\mathbf{v}_{x_i,i}\}_{i \in [\ell]}$ are distributed uniformly subject to the condition $\mathbf{d}_{x_i,i} + \mathbf{v}_{x_i,i} = \mathbf{y}_{x_i,i}$ for all $i \in [\ell]$, as the pads $\mathbf{v}_{0,i}$ and $\mathbf{v}_{1,i}$ provided by $\mathcal{F}_{\mathsf{ROT}}$ are random and independent of R's view, except with negligible probability.

4 Fixing the Consistency Check with SoftSpokenOT

As pointed out by Roy [26], the protocol from Fig. 7 is not as secure as we originally claimed, because Lemma 1 is incorrect and in certain cases a malicious receiver can break security. Roy presented a general protocol, called SoftSpokenOT, which addresses this issue, whilst also incorporating other improvements.

Protocol $\mathsf{DeROT}^{\kappa,\ell}$

1. The parties run $\mathsf{ROT}^{\kappa,\ell}$ with R inputting $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{\ell'})$, $\mathbf{x}_i \in \mathbb{F}_2^\kappa$, S receives $\{(\mathbf{v}_{0,i}, \mathbf{v}_{1,i})\}_{i\in[\ell]} \in \mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa$, and R receives $\{\mathbf{v}_{x_i,i}\}_{i\in[\ell]}$.
2. S sends $\{(\mathbf{d}_{0,i}, \mathbf{d}_{1,i})\}_{i\in[\ell]} = \{(\mathbf{v}_{0,i} + \mathbf{y}_{0,i}, \mathbf{v}_{1,i} + \mathbf{y}_{1,i})\}_{i\in[\ell]} \in \mathbb{F}_2^\kappa \times \mathbb{F}_2^\kappa$ to R.
3. R outputs $\{\mathbf{y}_{x_i,i}\}_{i\in[\ell]} = \{\mathbf{v}_{x_i,i} + \mathbf{d}_{x_i,i}\}_{i\in[\ell]}$.

Fig. 9. Derandomization protocol for random OT.

In Figure 10, to illustrate what we view as a straightforward way to fix our protocol, we show a simplified version of SoftSpokenOT for the case of 1-out-of-2 OT. We have not included various optimizations for reducing communication and computation.

Relation with SoftSpokenOT. [26] builds OT extension by first constructing subspace VOLE, which is a generalization of correlated OT, and then converting these into random OT. Subspace VOLE is defined over $\mathbb{F}_p^{k_\mathcal{C}}$, where p is prime and $\mathcal{C}$ is a linear code with dimension $k_\mathcal{C}$. Subspace VOLE is then converted into random 1-out-of-$p^{k_\mathcal{C}}$ OT through hashing.

Steps 1–4 of the **Extend** phase in Figure 10 correspond to building a passively secure subspace VOLE (as in [26, Fig. 7–8]) with $p = q = 2$ and $\mathcal{C}$ the repetition code with dimension $k_\mathcal{C} = 1$ and length $n_\mathcal{C} = \kappa$ (larger values of q are used for an optimization that reduces communication). The consistency check in Figure 10 is the same as the consistency check in [26, Fig. 9], with $k_C = 1$ and the linear universal hash function R given by

$$h_{\boldsymbol{\chi}} : \mathbb{F}_{2^s}^{m+1} \to \mathbb{F}_{2^s}, \quad \mathbf{x} \mapsto \sum_{j=1}^{m} x_j \chi_j + x_{m+1}$$

where $\boldsymbol{\chi} \in \mathbb{F}_{2^s}^m$ is the random seed defining the hash function. This is an $\mathbb{F}_2$-linear hash function, since multiplication in $\mathbb{F}_{2^s}$ is $\mathbb{F}_2$-linear, so defines a matrix $R \in \mathbb{F}_2^{s\times\ell'}$, which is $\mathbb{F}_2^\ell$-hiding [26, Definition 4.1] as required.

Instantiating the Hash Function. Care needs to be taken when choosing how to implement the hash function H. Subsequently to this paper, various works [12,20,26] pointed out that it is critical to include the index j as input to H, to prevent attacks. While a standard hash such as SHA-256 or SHA-3 should suffice, it is often more efficient to use a construction based on AES for fixed input lengths, since AES is commonly supported in hardware on modern CPUs. Guo et al. [12] showed that

$$H(j\|x) = \pi(\pi(x) \oplus j) \oplus \pi(x)$$

is a tweakable correlation robust hash function if π is modeled as an ideal cipher (which may in practice be implemented with fixed-key AES).

Protocol for $\mathsf{ROT}^{\kappa,\ell}$

The protocol uses an arbitrary stretch pseudorandom generator, PRG, and a hash function $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\kappa$, modeled as a random oracle.

Initialize:

1. R samples κ pairs of random κ-bit seeds, $\{(\mathbf{k}_0^i, \mathbf{k}_1^i)\}_{i=1}^{\kappa}$.
2. S samples a random $\Delta = (\Delta_1, \dots, \Delta_\kappa) \in \mathbb{F}_2^\kappa$.
3. The parties call $\kappa \times \mathsf{OT}_\kappa$ with inputs Δ and $\mathbf{k}_0, \mathbf{k}_1$.
4. S receives $\mathbf{k}_{\Delta_i}^i$ for $i = 1, \dots, \kappa$.

Extend: R inputs the choice bits $x_1, \dots, x_\ell \in \mathbb{F}_2$. Let $\ell' = \ell + s$, and assume that $s | \ell$.

1. R picks random $x_{\ell+1}, \dots, x_{\ell+s} \in \mathbb{F}_2$ and lets $\mathbf{x} = (x_1, \dots, x_{\ell'})$.
2. Expand $\mathbf{k}_i^0$ and $\mathbf{k}_i^1$, each using the next ℓ' bits from PRG, obtaining
$$\mathbf{t}_0^i = \mathsf{PRG}(\mathbf{k}_i^0) \in \mathbb{F}_2^{\ell'} \quad \text{and} \quad \mathbf{t}_1^i = \mathsf{PRG}(\mathbf{k}_i^1) \in \mathbb{F}_2^{\ell'}, \quad i = 1, \dots, \kappa.$$
so R knows $(\mathbf{t}_i^0, \mathbf{t}_i^1)$ and S knows $\mathbf{t}_{\Delta_i}^i$ for $i = 1, \dots, \kappa$.
3. R computes and sends $\mathbf{u}^i = \mathbf{t}_0^i + \mathbf{t}_1^i + \mathbf{x} \in \mathbb{F}_2^{\ell'}$, for $i = 1, \dots, \kappa$.
4. S computes
$$\mathbf{q}^i = \Delta_i \cdot \mathbf{u}^i + \mathbf{t}_{\Delta_i}^i \in \mathbb{F}_2^{\ell'}.$$
Write $\mathbf{t}^i = \mathbf{t}_0^i$, so that $\mathbf{q}^i = \mathbf{t}^i + \Delta_i \cdot \mathbf{x}$, for $i = 1, \dots, \kappa$.

Consistency check: Let $m = \ell/s$. We divide the ℓ' OTs into $m+1$ blocks of s bits, writing $\mathbf{x} = (\hat{x}_1, \dots, \hat{x}_{m+1}) \in \mathbb{F}_{2^s}^{m+1}$, and similarly $\mathbf{t}^i = (\hat{t}_1^i, \dots, \hat{t}_{m+1}^i) \in \mathbb{F}_{2^s}^{m+1}$, $\mathbf{q}^i = (\hat{q}_1^i, \dots, \hat{q}_{m+1}^i) \in \mathbb{F}_{2^s}^{m+1}$, for $i = 1, \dots, \kappa$.

1. S samples and sends $(\chi_1, \dots, \chi_m) \overset{\$}{\leftarrow} \mathbb{F}_{2^s}^m$.
2. R computes the following values over $\mathbb{F}_{2^s}$ and sends them to S
$$x = \sum_{j=1}^{m} \hat{x}_j \cdot \chi_j + \hat{x}_{m+1}, \quad t^i = \sum_{j=1}^{m} \hat{t}_j^i \cdot \chi_j + \hat{t}_{m+1}^i, \quad \text{for } i = 1, \dots, \kappa.$$
3. S computes
$$q^i = \sum_{j=1}^{m} \hat{q}_j^i \cdot \chi_j + \hat{q}_{m+1}^i \in \mathbb{F}_{2^s}$$
and checks that $q^i = t^i + \Delta_i \cdot x$, for all $i = 1, \dots, \kappa$. If any check fails, output Abort.

Transpose and randomize:

1. Let $\mathbf{q}_j$ denote the j-th row of the $\ell' \times \kappa$ bit matrix $[\mathbf{q}^1 | \dots | \mathbf{q}^\kappa]$ held by S, and similarly let $\mathbf{t}_j$ be the j-th row of $[\mathbf{t}_0^1 | \dots | \mathbf{t}_0^\kappa]$, held by R.
2. R outputs
$$\mathbf{v}_{x_j,j} = \mathsf{H}(j \| \mathbf{t}_j), \quad j \in [\ell].$$
3. S outputs
$$\mathbf{v}_{0,j} = \mathsf{H}(j \| \mathbf{q}_j) \quad \text{and} \quad \mathbf{v}_{1,j} = \mathsf{H}(j \| \mathbf{q}_j + \Delta), \quad j \in [\ell].$$

Fig. 10. A simple instantiation of SoftSpokenOT [26], which fixes the issue with the consistency check of Fig. 7

More efficient instantiations can be obtained by combining the ideal cipher with a universal hash function [6,26], reducing the cost to as little as one call to π for each call to H.

5 Implementation

In this section, we evaluate the efficiency of our random OT extension protocol. As was done in previous works [1,2], we tested the protocol in a standard LAN setting and a simulated WAN environment, using the Linux `tc` tool to create an average round-trip-time of 100 ms (with standard deviation 1 ms) and limit bandwidth to 50 Mbps (comparable with the setting reported by Asharov et al. [2]). We used computational security parameter $\kappa = 128$ and statistical security parameter $s = 64$ throughout, and instantiated the PRG with AES-128 in counter mode (using Intel AES-NI) and the random oracle H with SHA-1. $\mathcal{F}_{\mathsf{Rand}}$ is implemented using a standard hash-based commitment scheme, where both parties commit to and then open a seed, then the XOR of the two values is used to seed a PRG, which is UC-secure in the random oracle model.

Our implementation was written in C++ using the Miracl library for elliptic curve arithmetic in the base OTs, which were executed using the actively secure protocol of Peikert et al. [23]. All benchmarks were taken as an average of 20 runs on Intel Core i7-3770S 3.1 GHz processors with 8 cores and 32 GB of memory.

Implementation Optimizations. The correlation check stage of our protocol requires computing values of the form $\sum_i x_i \cdot y_i$ where $x_i, y_i \in \mathbb{F}_{2^\kappa}$. We used Intel PCLMUL instructions to efficiently compute carryless multiplications and then performed summations and the check itself in the polynomial ring (of length $2\kappa - 1$) to avoid having to do expensive reduction by the finite field polynomial.

As was done by Asharov et al. [1], we use Eklundh's algorithm for transposing the matrices T and Q during the COTe protocol in a cache-friendly manner, which makes the time spent in this stage less than 3% of the total runtime. Our implementation also supports multi-threading, making use of the 8 cores available on our test machines.

5.1 Comparison of Protocols

Table 2 shows the time taken for our implementation to compute 10 million OT extensions (excluding the base OTs) in a variety of settings. The one-directional setting is a traditional OT between a sender and a receiver, whilst in the bi-directional times, both parties perform both roles simultaneously (for a total of 20 million OTs). The bi-directional variant is often required for secure two-party and multi-party computation protocols, and over a LAN is much more efficient than performing the one-directional protocol twice, but less so in the WAN setting where communication is the bottleneck.

The passive protocol is just the standard IKNP extension (with the random OT communication optimization of Asharov et al. [1]), which is essentially our

Protocol	Comms. (MB)	LAN time (s)		WAN time (s)	
		One-dir.	Bi-dir.	One-dir.	Bi-dir.
Passive, IKNP (1T)	160MB	9.1037	12.5148	36.2319	66.2692
Passive, IKNP (8T)		3.3258	4.0827	28.4410	53.3977
Active, ours (1T)	160MB	9.5589	12.9461	36.2653	66.6558
Active, ours (8T)		3.3516	4.2020	28.4569	54.1157

Table 2. Random OT extension runtimes in seconds, using either 1 or 8 threads. The one-directional time is for 10 million OTs between a sender and receiver, whilst for the bi-directional time both parties are playing each role for a total of 20 million OTs.

protocol without the correlation check. In the LAN setting, the time difference between the active and passive protocols is less than 5%. The WAN times for the passive and active protocols are very similar, however it should be noted that there was more variation in our WAN experiments – computing 95% confidence intervals for these means in the table gives a variation of up to ±3%. This is probably mostly due to network variation and can be taken as evidence that our protocol has roughly the same performance as the passive IKNP extension. The total amount of data sent (in all protocols) is almost identical, due to the very low overhead of our correlation check. Compared with the reported timings for the protocol of Asharov et al. [2], our runtimes are much improved: their actively secure times are between 40% and 80% higher than their passive implementation, whilst ours almost match the efficiency of the passive protocol. (We do not directly compare figures due to the different benchmarking environments involved.)

Fig. 11 illustrates the performance of our protocol as the number of OTs computed varies, in both the WAN and LAN settings, tested in the one-directional and bi-directional modes of OT operation.

5.2 Profiling

Fig.12 presents profiling results for the main components of our protocol, run in a single thread creating 10 million OTs. It clearly demonstrates that the bottleneck of our protocol is communication from the IKNP extension phase, as was reported for the passive secure implementation of Asharov et al. [1]. The correlation check that we require for active security has a negligible impact on the runtime; the best way to further optimize our implementation in the LAN setting would be to target the hash function computations. The 'Other' section

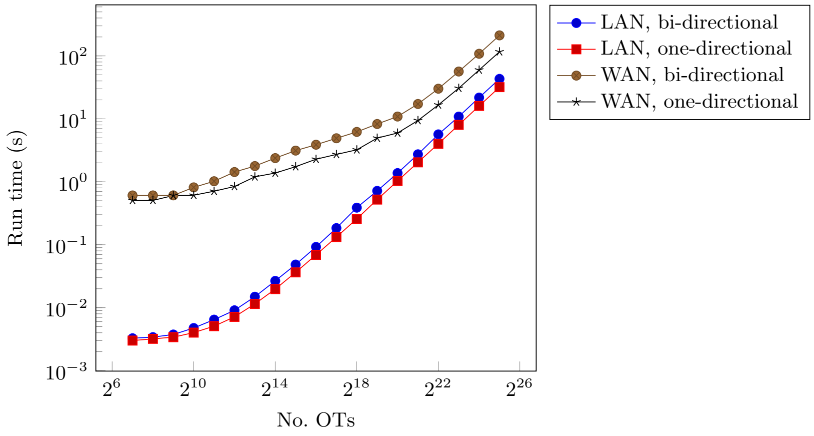

Fig. 11. Performance of our OT extension protocol for various numbers of OTs. Times exclude the base OTs.

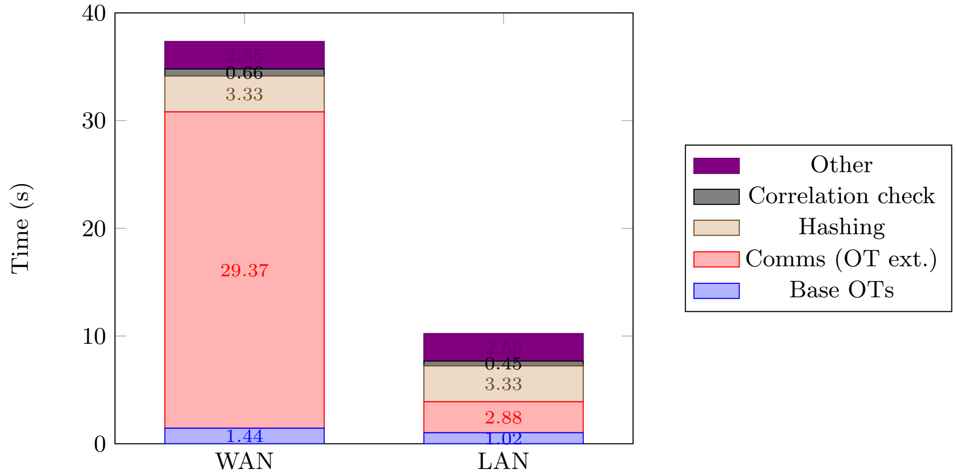

Fig. 12. Profiling results for running 10 million OTs in a single thread.

includes overhead from PRG computations, matrix transposition and allocating memory, which could also potentially be reduced a small amount.

6 Acknowledgments

We would like to thank Claudio Orlandi and Morten Bech for pointing out minor errors in the proof of Theorem 1, and the anonymous reviewers whose comments helped to improve the paper. This work has been supported in part by EPSRC via grant EP/I03126X.

References

1. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548. ACM, 2013.
2. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2015*, pages 673–701, 2015.
3. D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488. ACM, 1996.
4. G. Brassard, C. Crepeau, and J.-M. Robert. All-or-nothing disclosure of secrets. In A. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238. Springer Berlin Heidelberg, 1987.
5. 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. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/.
6. Y. L. Chen and S. Tessaro. Better security-efficiency trade-offs in permutation-based two-party computation. pages 275–304, 2021.
7. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.
8. 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 - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, pages 398–415, 2014.
9. C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 789–800, 2013.
10. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.
11. O. Goldreich and R. Vainish. How to solve any protocol problem - an efficiency improvement. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 73–86, 1988.
12. C. Guo, J. Katz, X. Wang, and Y. Yu. Efficient and secure multiparty computation from fixed-key block ciphers. pages 825–841, 2020.
13. R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 44–61. ACM, 1989.
14. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 145–161, 2003.
15. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer – efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591. Springer, 2008.

16. J. Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31, 1988.
17. V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 54–70, 2013.
18. E. Larraia. Extending Oblivious Transfer Efficiently or - How to get active security with constant cryptographic overhead. In *LATINCRYPT 2014 – Third International Conference on Cryptology and Information Security in Latin America*, 2014.
19. E. Larraia, E. Orsini, and N. P. Smart. Dishonest majority multi-party computation for binary circuits. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 495–512, 2014.
20. D. Masny and P. Rindal. Endemic oblivious transfer. pages 309–326, 2019.
21. J. B. Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. *IACR Cryptology ePrint Archive*, 2007:215, 2007.
22. 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 2012*, pages 681–700. Springer, 2012.
23. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 554–571, 2008.
24. B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 797–812, 2014.
25. M. O. Rabin. How to exchange secrets with oblivious transfer, 1981. Harvard University Technical Report 81.
26. L. Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 13-18, 2022, Proceedings*, 2022. https://eprint.iacr.org/2022/192.
27. S. Wiesner. Conjugate coding. *SIGACT News*, 15(1):78–88, Jan. 1983.
28. A. C. Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

A Proof of Lemma 2

Proof. Let $\rho_\kappa(n)$ be the rank of A and let $\mathbf{a}_1, \ldots, \mathbf{a}_\kappa \in \mathbb{F}_2^{\kappa+m}$ be the columns of A. Let E_i be the event that $\mathbf{a}_1, \ldots, \mathbf{a}_i$ are linearly independent, for $i = 1, \ldots, \kappa$. Note that $\Pr[\neg E_1] = 2^{-(\kappa+m)}$ and for $i > 1$:

$$\Pr[\neg E_i \mid E_{i-1}] = \frac{2^{i-1}}{2^{\kappa+m}} = 2^{i-1-\kappa-m}$$

since $\neg E_i$ happens if and only if $\mathbf{a}_i$ lies in the space spanned by $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$, of size 2^{i-1}. Now by a union bound it follows that

$$\Pr[\rho_\kappa(\kappa+m) < \kappa] = \Pr[\neg E_\kappa] \leq \sum_{i=1}^{\kappa} \Pr[\neg E_i \mid E_{i-1}] = \sum_{i=1}^{\kappa} 2^{i-1-\kappa-m} \leq 2^{-m}.$$

□