Batching Base Oblivious Transfers

Ian McQuoid* Mike Rosulek* Lawrence Roy*

May 24, 2021

Abstract

Protocols that make use of oblivious transfer (OT) rarely require just one instance. Usually a batch of OTs is required — notably, when generating base OTs for OT extension. There is a natural way to optimize 2-round OT protocols when generating a batch, by reusing certain protocol messages across all instances. In this work we show that this batch optimization is error-prone. We catalog many implementations and papers that have an incorrect treatment of this batch optimization, some of them leading to catastrophic leakage in OT extension protocols.

We provide a full treatment of how to properly optimize recent 2-round OT protocols for the batch setting. Along the way we show several performance improvements to the OT protocol of McQuoid, Rosulek, and Roy (ACM CCS 2020). In particular, we show an extremely simple OT construction that may be of pedagogical interest.

1 Introduction

Oblivious transfer (OT) is a fundamental primitive for cryptographic protocols. It is well-known that OT cannot be constructed in a black-box way from symmetric-key primitives [IR90]. Nevertheless, it is possible to generate a large number of OTs from symmetric-key primitives and *a small number of "base OTs"*, thanks to an idea called **OT extension** [Bea96]. OT extension to generate a polynomially large number N of OTs means that the *marginal cost* of OTs involves only cheap symmetric-key operations. Modern OT extension protocols [IKNP03, KK13, ALSZ13, KOS15] can generate millions of OTs per second.

OT extension protocols require κ (*e.g.*, 128) base OTs, and yet most base-OT protocols in the literature are described in terms of a single OT instance. Obviously any single-instance OT protocol can be invoked κ times to produce base OTs; however, this overlooks the possibility of optimizations for the batch setting. In this work we provide a full treatment of the batch setting for recent leading OT protocols.

1.1 Overview of Our Results

There is a natural way to optimize certain 2-round OT protocols for the batch setting. When the OT sender is first to speak, it is natural to reuse their protocol message for all OT instances in the batch. We call this method **naïve batching**.

We show that naïve batching is very often insecure. Not only does naïve batching fail to achieve an appropriate security notion, it is also demonstrably unsuitable as the base OTs for certain OT extension protocols. Specifically, we show a serious attack on the 1-out-of-N OT

*Oregon State University, {mcquoidi,rosulekm,royl}@oregonstate.edu. Third author is partially supported by a DoE CSGF Fellowship.

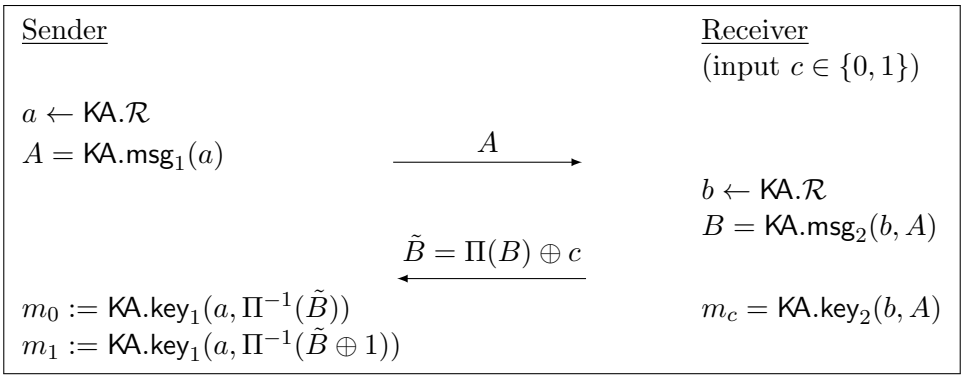

Figure 1: Our simple 1-of-2 random OT protocol. $\Pi^{\pm}$ is an ideal permutation and KA is a 2-message key agreement whose "B-messages" are pseudorandom bit strings.

extension protocol of Orrù, Orsini, and Scholl [OOS17], when its base OTs are generated with naïve batching. Unfortunately, we find naïve (or similarly improper) batching appearing in several protocol libraries [Rin, CMR, Kel20, Sma] and in several papers [CO15, HL17, CSW20].

We then give a complete treatment of how to correctly optimize leading OT protocols for the batch setting. Fortunately it is simple and cheap to fix naïve batching, although the complete security analysis requires care. We show how to correctly optimize the recent OT protocol of McQuoid, Rosulek, and Roy [MRR20] for the batch setting. As we show, the Masny-Rindal protocol [MR19] is a special case of the McQuoid-Rosulek-Roy protocol, and our analysis applies to that protocol as well. A comparison of our protocol to existing work is shown in Table 1.

Finally, we present several new improvements to the McQuoid-Rosulek-Roy (MRR) protocol. Their OT protocol is actually an oblivious PRF protocol: the sender learns a pseudorandom function F, the receiver chooses point x and learns $F(x)$. Such a protocol yields 1-out-of-n [random] OT by letting $F(1), \ldots, F(n)$ be the n OT values, of which the receiver can learn only one. Their protocol supports F with domain $\{0,1\}^*$ (*i.e.*, 1-out-of-n OT for *any* n).

- The MRR protocol revolves around an object called a **programmable-once public function (POPF)**. A POPF with domain $[n]$ leads to a protocol for 1-out-of-n endemic OT. MRR describe a new POPF with domain $\{0,1\}^*$, leading to their OPRF protocol. But this is overkill for the case of 1-out-of-2 OT, which is all that is needed for OT extension. We show several improved POPF constructions for small domains (such as $n = 2$). One particularly interesting POPF is in the ideal random permutation model[1] and is inspired by the Even-Mansour block cipher construction [EM93]. When we instantiate MRR with this new POPF, we obtain an endemic OT protocol that is incredibly simple to describe. The protocol is not only efficient, but may be have pedagogical value as well. See Figure 1.

- The MRR protocol constructs endemic OT from a POPF and a key agreement (KA) protocol. These two components must be compatible, and in [MRR20] this was achieved at the cost of some nontrivial additional overhead — either hash-to-curve operations or Elligator [BHKL13] encoding steps. In this work, we suggest an alternative based on a trick due to Möller [Möl04]. Möller-DHKA avoids complicated encodings and the need for hash-to-curve operations in the protocol. It also avoids curve point addition, allowing us to use Montgomery Ladders to multiply, which are more efficient. It requires doubling the length of the sender's protocol

[1] Like the random oracle model, all parties have access to a random permutation on $\{0,1\}^{2\kappa}$, *and its inverse!*

Scheme	Assumption	Setup	Flows	Exp (sender/receiver)	Comm (sender/receiver)
SimplestOT [CO15]	Gap-CDH	PRO	2	1f $(m+1)$v / mf mv	$1\mathbb{G}$ / $m\mathbb{G}$
BlazingOT [CSW20]	CDH	ORO	3	1f $(m+1)$v / mf mv	$m\kappa + 1\mathbb{G}$ / $2\kappa + m\mathbb{G}$
EndemicOT [MR19]	DDH	PRO	2	$2m$f $2m$v / mf mv	$2m\mathbb{G}$ / $2m\mathbb{G}$
EndemicOT [MR19]	iDDH	PRO	1	mf $2m$v / mf mv	$m\mathbb{G}$ / $2m\mathbb{G}$
Ours (MasnyRindal)	ODH	PRO	1	2fM $2m$vM / mfM mvM	$2\mathbb{G}$ / $2m\mathbb{G}$
Ours (EKE/EvenMansour)	ODH	IC	1	2fM $2m$vM / mfM mvM	$2\mathbb{G}$ / $m\mathbb{G}$
Ours (Feistel)	ODH	PRO	1	2fM $2m$vM / mfM mvM	$2\mathbb{G}$ / $m\mathbb{G}$

Table 1: Comparison of m-instance random 1-of-2 OT protocols. "Exp" denotes exponentiations (f = fixed-base, v = variable-base, fM = fixed-base Montgomery, vM = variable-base Montgomery). "Comm" denotes communication ($\mathbb{G}$ = one group element). PRO = programmable random oracle; ORO = observable random oracle; IC = ideal cipher.

message; however, in the batch setting it is exactly this sender's message that is reused across all OT instances in the batch, so the effect of doubling its size is minimal.

Finally, we show how our batch OT protocol can be used as the base OTs in **2-round** OT extension.

2 Preliminaries

2.1 Endemic OT

We use the security definitions for universally composable OT suggested by [MR19], which are a convenient middle-ground between random OT and chosen-message OT. An OT protocol results in outputs r_0, r_1 for the sender and r_c for the receiver (who has choice bit c). In **endemic OT**, a corrupt party may choose their own OT outputs, and all other OT outputs are chosen uniformly by the functionality. Hence, a corrupt sender can choose both r_0 and r_1. A corrupt receiver can choose r_c and the functionality will ensure that r_{1-c} is uniform.

As shown in [MR19], OT extension protocols are secure if the base OTs satisfy this notion of endemic OT.

In this work we consider the batch setting, in which the parties wish to generate a batch of n OTs at once. In Figure 2 we present a functionality that realizes n instances of OT with endemic security.

3 Problems With Naïve Batching

3.1 Naïve Batching

Consider an abstract 2-round protocol for (endemic) OT, with the following syntax:

Functionality $\mathcal{F}_{\text{batchEOT}}$

The functionality $\mathcal{F}_{\text{batchEOT}}$ is parameterized by the length of the OT strings ℓ and the number n of OTs in the batch. It interacts with two parties, a sender S and a receiver R via the following queries:

On input $(\text{READY}, (\tilde{r}_{1,0}, \tilde{r}_{1,1}, \ldots, \tilde{r}_{n,0}, \tilde{r}_{n,1}))$ from S, with $\tilde{r}_{i,c} \in \{0,1\}^\ell$:

- If S is corrupt, and there has been no previous READY command from S, then internally record $r_{i,c} = \tilde{r}_{i,c}$ for all $i \in [n]$, $c \in \{0,1\}$. Otherwise do nothing.

On input $(\text{READY}, (c_1, \ldots c_n) \in \{0,1\}^n, (\tilde{r}_1, \ldots, \tilde{r}_n))$ from R, with $\tilde{r}_i \in \{0,1\}^\ell$:

- Do nothing if there has been a previous READY query from R.
- Internally record $(c_1, \ldots c_n)$
- If R is corrupt, then internally record $r_{i,c_i} = \tilde{r}_i$ for each $i \in [n]$.

After receiving READY queries from both S and R:

- For all $i \in [n], c \in \{0,1\}$, if $r_{i,c}$ is not already defined, then sample $r_{i,c} \leftarrow \{0,1\}^\ell$.
- Output $(r_{1,c_1}, \ldots, r_{n,c_n})$ to R and $((r_{1,0}, r_{1,1}), \ldots, (r_{n,0}, r_{n,1}))$ to S.

Figure 2: Batch Endemic 1-out-of-2 Oblivious Transfer functionality $\mathcal{F}_{\text{batchEOT}}$. Adapted from the endemic OT functionality of [MR19].

Sender		Receiver (input $c \in \{0,1\}$)
$s_S \leftarrow \{0,1\}^\kappa$		
$m_S = \mathsf{OT.msg}_S(s_S)$	$\xrightarrow{m_S}$	
		$s_R \leftarrow \{0,1\}^\kappa$
		$m_R = \mathsf{OT.msg}_R(s_R, m_S, c)$
	$\xleftarrow{m_R}$	
$(r_0, r_1) = \mathsf{OT.out}_S(s_S, m_R)$		$r_c = \mathsf{OT.out}_R(s_R, m_S, c)$

In such a protocol, the sender's message m_S is clearly independent of the receiver's influence. In many protocols m_S is also a message from a KA protocol, and it is well-known that in many KA constructions a single KA message can be reused for many KA instances. These observations suggest the following optimization for generating a batch of n OTs:

Sender		Receiver (inputs $\{c_i\}_{i\in[n]}$)
$s_S \leftarrow \{0,1\}^\kappa$		
$m_S = \mathsf{OT.msg}_S(s_S)$	$\xrightarrow{m_S}$	
		for $i \in [n]$:
		$s_{R,i} \leftarrow \{0,1\}^\kappa$
		$m_{R,i} = \mathsf{OT.msg}_R(s_{R,i}, m_S, c_i)$
	$\xleftarrow{m_{R,1}, \ldots, m_{R,n}}$	
for $i \in [n]$:		for $i \in [n]$:
$(r_{i,0}, r_{i,1}) = \mathsf{OT.out}_S(s_S, m_{R,i})$		$r_{i,c_i} = \mathsf{OT.out}_R(s_{R,i}, m_S, c_i)$

We call this protocol transformation **naïve batching**. Note that this protocol reuses $\mathsf{OT.out}_R$ in such a way that disallows for internal domain separation.

Lemma 1. *Naïve batching does **not** securely realize batch endemic OT (Figure 2).*

Proof. The attack is simple: a corrupt receiver simply sends $m_{R,1} = \cdots = m_{R,n}$. As a result, the sender must compute $(r_{1,0}, r_{1,1}) = \cdots = (r_{n,0}, r_{n,1})$. There is no way for the simulator to influence the sender's output in this way in the ideal model, hence this constitutes an attack. □

Why not trivially patch this attack? The attack is for the receiver to send the same OT response for all instances. We could simply tell the sender to abort if it receives any repeated OT responses.

However, the simple attack that we have described is only the tip of the iceberg. In all of the 2-round OT protocols that we consider, a corrupt receiver can induce more complicated correlations among the OT values. For example, a receiver can act honestly in the first OT instance to learn $r_{1,0}$. Then $r_{1,1}$ is unknown to the receiver. But there is a more sophisticated strategy for the receiver to force the ratio $r_{1,1}/r_{2,0}$ to be a certain value. (The details of this strategy depend on the details of a specific base OT protocol, so we defer them to Appendix A.)

Based on this kind of attack, one might wish to weaken the endemic OT functionality. Why not allow the simulator to specify these kinds of correlations in the ideal model? Even this will not work, because the attack is *perfectly* indistinguishable from honest behavior by the receiver. Thus, there is simply no way for the simulator to distinguish this kind of an attack (where the receiver must learn $r_{1,1}/r_{2,0}$) vs. honest behavior (where the receiver must learn $r_{2,0}$).

For these reasons, we believe there is no way to closely capture the security of naïve batching in a UC ideal functionality.

3.2 Implications for OT Extension

Since the main application for batch OTs is as base OTs for OT extension, it is natural to wonder whether the simple attack above jeopardizes the security of OT extension. It has been established that OT extension can be securely realized from base OTs with weakened security. For example, [CSW20] show that certain input-dependent aborts in the base OTs do not harm the security of OT extension.

We show that out simple attack on naïve batching indeed compromises security of some OT extension protocols. Specifically, we consider the protocol of Orrù, Orsini, and Scholl (OOS) [OOS17]. This OT extension protocol generates many instances of 1-out-of-2^t OT, where in each one the sender obtains $r_1, \ldots, r_{2^t}$ and the receiver learns only r_c, where c is an input. It will be convenient to consider c to be an element of $\{0,1\}^t$ in the natural way.

The OOS protocol is secure when the base OTs securely realize endemic batch OT; see [MR19] for details. However, it loses security when using naïve batching to generate its base OTs.

Lemma 2. *The OOS protocol [OOS17] is demonstrably insecure when its base OTs are instantiated via naïve batching.*

Proof. The complete details of OOS can be found in [OOS17]. We sketch the relevant details of their protocol here.

Let Alice be the OOS sender (with no inputs) and Bob be the OOS receiver (with choice value $c_i \in \{0,1\}^t$ for the ith OT instance). The protocol proceeds as follows:

- The parties run ℓ base OT instances, with Alice acting as receiver and Bob acting as sender. Bob obtains base-OT outputs $(k_{1,0}, k_{1,1}), \ldots, (k_{\ell,0}, k_{\ell,1})$. Alice's inputs and outputs are not relevant here.

- When extending to n OTs, Bob constructs two $n \times \ell$ matrices K and R as follows:

  - The jth column of K is $\mathsf{PRG}(k_{j,0}) \oplus \mathsf{PRG}(k_{j,1})$.
  - The ith row of R is $C(c_i)$ where $C : \{0,1\}^t \to \{0,1\}^\ell$ is a suitable binary error correcting code (the details of which are not relevant here).

  Bob sends $K \oplus R$ to Alice.

These details of OOS are enough to understand the attack. A corrupt Alice will attack the base OTs (in the role of OT receiver as above) so that all $k_{i,0}$'s are the same and all $k_{i,1}$'s are the same. As a result, every column of K is identical. In other words, every *row* of K is either 0^ℓ or 1^ℓ.

Then the ith row of Bob's matrix $K \oplus R$ is either $C(c_i)$ or its complement. This means that if $c, c' \in \{0,1\}^t$ are any two choices for Bob whose codewords are not bitwise complements of each other, then Alice can distinguish between Bob having choice c vs c' in each extended OT. For some choices of C, learning $C(x)$ up to complement uniquely reveals x. This attack results in almost complete leakage of Bob's private input. □

What if C is a repetition code? C is a binary error-correcting code, the simplest of which is the repetition code $C : \{0,1\} \to \{0,1\}^\ell$. This corresponds to the case of $t = 1$, and hence 1-out-of-2 OT extension. Specifically, instantiating OOS with a repetition code collapses it to the Keller-Orsini-Scholl 1-out-of-2 OT extension protocol (KOS) [KOS15].

In this case the only two codewords are 0^ℓ and 1^ℓ. Since these are bitwise complements of one another, it is not clear that our attack leads to any security problems. The rows of matrix R (encoding Bob's private input) are masked by either 0^ℓ or 1^ℓ, depending on a bit that is unknown to Alice. We are not sure whether a more sophisticated attack on the base OTs (even for a specific naïvely batched OT) can break KOS OT extension.

3.3 Problematic Batching Found in the Wild

Looking ahead, the fix for naïve batching is simple and essentially free (although the security analysis of the fix requires some care, as we show in the next sections). In Diffie-Hellman-based OT protocols, the OT outputs r_0, r_1 are computed by taking a (random oracle) hash of a Diffie-Hellman value. The fix is to include the OT index in that key derivation — *i.e.*, instead of $r_0 = H(sid, g^{ab})$, use $r_0 = H(sid, g^{ab}, \underline{i})$ in the ith OT instance in the batch. That way, even if all g^{ab} values are identical (or correlated strangely), the final OT values are independently random.

Given that both the attack and the fix are so simple, one may wonder whether this problem is well-known. In fact, we found problems related to OT batching in many libraries that implement malicious-secure OT extension.[2] We focus on the implications for the overall OT extension protocols, which are minor in most cases. However, the consequences would be more severe for developers that directly access the base-OT functionalities of these libraries.

- The `libote` OT extension library [Rin] implements Masny-Rindal [MR19] base OTs and applies naïve batching. The original Masny-Rindal paper considers only the single-instance

[2] We notified the maintainers of these libraries about the issues and the suggested fix. By the time of writing, all maintainers have either already fixed or planned to fix their handling of batch OTs.

setting and does not discuss security of the batch setting under naïve batching. In some configurations, the `libote` implementation of OOS indeed uses these naïvely batched base OTs, thus falling victim to our attack. Other configurations use a hybrid approach, first naïvely batching 128 base OTs, then using KOS to extend to 512 OTs, and using those 512 OTs as base for OOS. As mentioned above, we are not aware of any explicit attack on KOS extension, but our observations merely raise some concerns about its security with naïvely batched OTs.

- The `swanky` MPC library [CMR] implements the Chou-Orlandi protocol and reuses the sender's message, but uses good domain separation in key derivation.[3] However, it allows the sender's protocol message to be reused across several batches, while the domain separation is local to the batch! In other words, parties could execute two batches of OTs, and the receiver could cause the batches to produce identical outputs, by replaying its protocol messages.

  In this library's implementation of OT extension, they first apply the transformation in [MR19] from endemic OT to uniform-message OT on the base OTs. This prevents the receiver from forcing OT extension to operate on identical base OTs. If not for this additional step, even KOS OT extension would leak information across different batches. As it is, only the XOR of PRG seeds is leaked under our attack on naïve batching, which is unlikely to lead to a concrete attack.

- The `mp-spdz` [Kel20] and `scale-mamba` [Sma] library implementations of OT use naïve batching of Chou-Orlandi base OTs. These libraries implement only KOS and not OOS, and therefore we know of no concrete attack against their OT extension.

We have also identified problematic handling of OT batching in several papers:

- The Chou-Orlandi OT protocol [CO15] explicitly considers the batch setting and uses naïve batching to achieve it. As such, the protocol as written is not suitable as the base OT for certain OT extensions.

- Since security flaws (unrelated to batching) were discovered in the Chou-Orlandi protocol, several works have attempted to address and repair them. Of those works, both [HL17] and [CSW20] explicitly consider the batch setting. The paper of Hauck & Loss [HL17] maintains the naïve batching of the original.

- The "Blazing OT" construction of Canetti, Sarkar, and Wang [CSW20] does not technically use naïve batching, since it introduces a joint consistency check across all instances in the batch. However, the key derivation in their base OTs does *not* include the OT index. This means that the attack in Lemma 1 has the intended effect: causing all OT instances to give identical output. The paper only considers a combined protocol with batched Chou-Orlandi base OTs and KOS OT extension, and as such we are not aware of an explicit attack on their final OT extension protocol. However, their security analysis does not seem to acknowledge the possibility of all base OTs giving identical outputs.

We found one instance of totally correct batching, in the implementation of Blazing OT in `emp-toolkit` [WMK16], despite correct batching not being described explicitly in the Blazing OT paper.

[3] The authors explicitly justify their correct key derivation as a bug in the Chou-Orlandi paper, and reference the attack in which all base OTs generate identical output. See `chou_orlandi.rs`.

4 Properly Batching OTs

In this section we describe how to repair naïve batching. We focus on the McQuoid-Rosulek-Roy (MRR) protocol [MRR20] since it subsumes the Masny-Rindal protocol, and the Chou-Orlandi protocol does not achieve UC security. As we saw, the main problem is that a corrupt receiver can force correlations among the OT outputs in different instances — even causing some OT values to be equal. The solution is to enforce "domain separation" among the different instances. Intuitively, parties should hash each instance's OT outputs under a random oracle, with domain separation (*i.e.*, include the index of that instance in the hash).

However, proving the security of this change requires some care. For example, we cannot prove security merely from the single-instance security of the OT protocol, since the single-instance protocol is not being used correctly. Instead, we must use some known structure of the protocol. The McQuoid-Rosulek-Roy (MRR) protocol derives its outputs from its underlying KA protocol, and we require stronger properties from that KA. The KA must accept an extra "tag" argument, so that even if the KA messages are identical, the resulting keys will be different under different tags.

4.1 Tagged KA

A **tagged KA** is identical in syntax to a traditional KA, except that the $\mathsf{KA.key}_1$ and $\mathsf{KA.key}_2$ algorithms take an additional tag argument. Correctness is that for all $a, b \in \mathsf{KA}.\mathcal{R}$ and all tags τ:

$$\mathsf{KA.key}_1(a, \mathsf{KA.msg}_2(b, \mathsf{KA.msg}_1(a)), \tau) = \mathsf{KA.key}_2(b, \mathsf{KA.msg}_1(a), \tau)$$

Looking ahead to our batch OT protocol, we will let the tag τ be the index of the OT instance (*e.g.*, OT instance 1, 2, 3, ...).

Intuitively, we will require that KA outputs under different tags appear indpendently random. This should hold not only when the KA protocol messages are identical, but also when the KA messages (*e.g.*, $\mathsf{KA.msg}_2$) are correlated, since we previously observed (Section 3) that the adversary could induce arbitrary correlations across OT/KA instances.

Definition 3. *A tagged KA protocol is* ***tag-non-malleable*** *if a session with tag τ^* is secure, even against an eavesdropper that has oracle access to $\mathsf{KA.key}_1(a, \cdot, \cdot)$, provided the eavesdropper never queries the oracle on tag τ^*. Formally, the following distributions are indistinguishable, for all τ^* and every PPT $\mathcal{A}$ that never queries its oracle with second argument τ^*:*

$a, b \leftarrow \mathsf{KA}.\mathcal{R}$	$a, b \leftarrow \mathsf{KA}.\mathcal{R}$
$M_1 = \mathsf{KA.msg}_1(a)$	$M_1 = \mathsf{KA.msg}_1(a)$
$M_2 = \mathsf{KA.msg}_2(b, M_1)$	$M_2 = \mathsf{KA.msg}_2(b, M_1)$
$K = \mathsf{KA.key}_1(a, M_2, \tau^*)$	$K \leftarrow \mathsf{KA}.\mathcal{K}$
return $\mathcal{A}^{\mathsf{KA.key}_1(a,\cdot,\cdot)}(M_1, M_2, K)$	return $\mathcal{A}^{\mathsf{KA.key}_1(a,\cdot,\cdot)}(M_1, M_2, K)$

Like [MRR20], we also require the KA protocol to satisfy the following randomness property:

Definition 4. *A key agreement protocol has* ***strongly random responses*** *if the honest output of $\mathsf{KA.msg}_2$ is indistinguishable from random, even to an adversary who (perhaps maliciously) gener-*

ated M_1. Formally, for all polynomial time $\mathcal{A}$, the following distributions are indistinguishable:

$$\boxed{\begin{array}{l}(M_1, state) \leftarrow \mathcal{A}() \\ b \leftarrow \mathsf{KA}.\mathcal{R} \\ M_2 = \mathsf{KA}.\mathsf{msg}_2(b, M_1) \\ \text{return } (state, M_2)\end{array}} \qquad \boxed{\begin{array}{l}(M_1, state) \leftarrow \mathcal{A}() \\ \\ M_2 \leftarrow \mathsf{KA}.\mathcal{M} \\ \text{return } (state, M_2)\end{array}}$$

4.2 Programmable-Once Public Functions

The McQuoid-Rosulek-Roy protocol uses a primitive called programmable-once public functions (POPFs). We introduce definitions for POPF here, which slightly differ from the original definitions. We have specialized the definitions for the case of 1-out-of-2 OT[4] — [MRR20] define POPFs in a way that is useful for 1-out-of-N OT (with exponential N) and also password-authenticated key exchange. In the original POPF definitions, a simulator simulated the random oracle setup in the service of a single POPF instance; in our batch setting there will be many POPF instances, thus we must adapt the definitions to explicitly allow simulation of multiple POPFs in a non-interfering way.

Definition 5. *A **1-weak random oracle** is a function $F\colon \mathcal{N} \to \mathcal{O}$ such that the following two distributions are indistinguishable,*

$$\boxed{\begin{array}{l}x \leftarrow \mathcal{N} \\ y := F(x) \\ \text{return } x, y\end{array}} \qquad \boxed{\begin{array}{l}x \leftarrow \mathcal{N} \\ y \leftarrow \mathcal{O} \\ \text{return } x, y\end{array}}$$

when the adversary does not have access to F other than through these experiments.

Note that F is only allowed to be used once this definition. This makes it an extremely weak property — it's even satisfied by universal hashes.

Definition 6 (Syntax)**.** *A **batch 2-way programmable-once public function (batch 2-POPF)** consists of algorithms:*

- $\mathsf{Eval} : \mathcal{M} \times \{0,1\} \to \mathcal{N}$
- $\mathsf{Program} : \{0,1\} \times \mathcal{N} \to \mathcal{M}$

Both algorithms access some local setup $\mathbb{H}$ — depending on the instantiation, $\mathbb{H}$ could consist of common reference strings, random oracles, ideal ciphers, etc. All parties (adversaries) may access the setup directly as well, although it is local to a single instance of the batch 2-POPF. The setup may be stateful (e.g., the "lazy" formulation of a random oracle, which samples outputs on the fly).

A 2-POPF must also include alternative local setups, which are used in different security definitions:

- *$\mathbb{H}_{\mathsf{HSim}}$ must provide the same interface as $\mathbb{H}$ as well as an additional method $\mathsf{HSim}\colon \mathcal{N} \times \mathcal{N} \to \mathcal{M}$.*

[4] All of the POPFs in this paper have straightforward generalizations to the 1-out-of-N case, for polynomial N, and some to exponential N as well, but we restrict ourselves to the 1-out-of-2 case for simplicity.

- $\mathbb{H}_{\mathsf{Extract}}$ *must provide the same interface as* $\mathbb{H}$ *as well as an additional method* $\mathsf{Extract}\colon \mathcal{M} \to \{0,1\}$. $\mathsf{Extract}$ *must not modify the private state of* $\mathbb{H}_{\mathsf{Extract}}$.

We write $\mathcal{A}^{\mathbb{H}}$ to denote an algorithm $\mathcal{A}$ with oracle access to all methods provided by the setup $\mathbb{H}$.

Definition 7 (Correctness)**.** *A batch 2-POPF satisfies* ***correctness*** *if* $\mathsf{Eval}(\phi, x^*) = y^*$ *with all but negligible probability, whenever* $\phi \leftarrow \mathsf{Program}(x^*, y^*)$.

Definition 8 (Security)**.** *A batch 2-POPF is* ***secure*** *if it satisfies the following properties:*

1. **Indistinguishable Local Setups:** *The local setups* $\mathbb{H}$, $\mathbb{H}_{\mathsf{HSim}}$ *and* $\mathbb{H}_{\mathsf{Extract}}$ *all implement a common interface. The setups must be indistinguishable to an adversary that only queries on this interface. Formally, if* $\mathcal{A}$ *is a polynomial-time algorithm that only queries its setup on the interface of* $\mathbb{H}$ *then the following probabilities are negligibly close:*

$$\Pr[\mathcal{A}^{\mathbb{H}}() = 1]; \qquad \Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{HSim}}}() = 1]; \qquad \Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{Extract}}}() = 1]$$

2. **Honest Simulation:** *Any* ϕ *that is generated honestly as* $\phi \leftarrow \mathsf{Program}(x^*, y^*)$*, with* y^* *chosen uniformly, is indistinguishable from* ϕ *generated via the* HSim *algorithm of* $\mathbb{H}_{\mathsf{HSim}}$*. Since* HSim *does not have a "preferred" input* x^**, this establishes that an honestly generated* ϕ *hides the* x^* *on which it was programmed.*

   Formally, define the following functions:

$\underline{\text{REAL\_PHI}(x^* \in \{0,1\}, \mathfrak{D})}$:
$(s, y^*) \leftarrow \mathfrak{D}$
$\phi \leftarrow \mathsf{Program}(x^*, y^*)$
$r_0 := \mathsf{Eval}(\phi, 0)$
$r_1 := \mathsf{Eval}(\phi, 1)$
return s, ϕ, r_0, r_1

$\underline{\text{SIM\_PHI}(x^* \in \{0,1\}, \mathfrak{D})}$:
$(s, y^*) \leftarrow \mathfrak{D}$
$r_{x^*} := y^*$
$r_{1-x^*} \leftarrow \mathcal{N}$
$\phi \leftarrow \mathsf{HSim}(r_0, r_1)$
return s, ϕ, r_0, r_1

   Then for all polynomial time $\mathcal{A}$*,*

$$\Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{HSim}}, \text{REAL\_PHI}}() = 1] - \Pr[\mathcal{A}^{\mathbb{H}_{\mathsf{HSim}}, \text{SIM\_PHI}}() = 1]$$

   is negligible. Here we restrict $\mathcal{A}$ *to always query with* $\mathfrak{D}$ *a distribution over* $\{0,1\}^* \times \mathcal{N}$ *such that the marginal distribution of* y^* *is indistinguishable from the uniform distribution over* $\mathcal{N}$*. The other component* s *appears for technical reasons; the reader can think of it as the coins used to sample* y^**.*

   Note that SIM_PHI *calls the* HSim *method of the local setup, and that* $\mathcal{A}$ *may even query the* HSim *method (both the real and ideal experiments use* $\mathbb{H}_{\mathsf{HSim}}$*).*

3. **Uncontrollable Outputs:** *For any* ϕ *generated by the adversary, the* $\mathsf{Extract}$ *method of* $\mathbb{H}_{\mathsf{Extract}}$ *can identify an input* x^* *such that the adversary has no control over* $\mathsf{Eval}(\phi, 1 - x^*)$*. We say that* $\mathsf{Eval}(\phi, 1 - x^*)$ *is beyond the adversary's control if* $F(\mathsf{Eval}(\phi, 1 - x^*))$ *is indistinguishable from random, for any 1-weak-RO* F*.*[5]

[5]There are 1-weak ROs whose outputs can be distinguished from random when inputs are chosen in a certain adversarial way. Hence, requiring the RO outputs to remain random is a way of requiring that these values are not chosen in an adversarial way.

Formally, the following distributions must be indistinguishable for all polynomial-time $\mathcal{A}_1, \mathcal{A}_2$ and all 1-weak-RO F:

$(\phi, state) \leftarrow \mathcal{A}_1^{\mathbb{H}_{\mathsf{Extract}}}()$ $x^* := \mathsf{Extract}(\phi)$ $r := F(\mathsf{Eval}(\phi, 1 - x^*))$ return $\mathcal{A}_2^{\mathbb{H}_{\mathsf{Extract}}}(state, r)$	$(\phi, state) \leftarrow \mathcal{A}_1^{\mathbb{H}_{\mathsf{Extract}}}()$ $r \leftarrow \mathcal{N}$ return $\mathcal{A}_2^{\mathbb{H}_{\mathsf{Extract}}}(state, r)$

As above, the left distribution calls the $\mathsf{Extract}$ method of the $\mathbb{H}_{\mathsf{Extract}}$ setup, and the adversary may query this method as well. Note that $\mathcal{A}$ does not have any access to F beyond the one call provided by this experiment.

The reader may be curious why we forced y^* to be sampled inside Honest Simulation, instead of letting the adversary choose it like in [MRR20]. The answer is that otherwise an ideal cipher would not be a POPF. An adversary could have already run $\mathsf{Program}(0, y^*)$ earlier, and because for each x there is a bijection between values of y and ϕ, a call to $\mathsf{HSim}(y^*, r_1)$ would be forced to return the same ϕ as before. Ideal ciphers were used as a motivating example for POPFs in [MRR20], so this is clearly a mistake. Ideal ciphers satisfy our new definition (Section 5.1).

4.3 The Batch OT Protocol

Sender		Receiver (with input $\{c_i\}_{i\in[n]}$)
$a \leftarrow \mathsf{KA}.\mathcal{R}$		
$m_S = \mathsf{KA.msg}_1(a)$	$\xrightarrow{m_S}$	for $i \in [n]$: $b_i \leftarrow \mathsf{KA}.\mathcal{R}$ $m_{R,i} := \mathsf{KA.msg}_2(b_i, m_S)$ $\phi_i := \mathsf{Program}(c_i, m_{R,i})$
for $i \in [n]$:	$\xleftarrow{\phi_1, \ldots, \phi_n}$	
for $j \in \{0,1\}$:		for $i \in [n]$:
$r_{i,j} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, j), i \parallel j)$		$r_{i,c_i} = \mathsf{KA.key}_2(b_i, m_S, i \parallel c_i)$

Figure 3: Our n-batch 1-of-2 Oblivious Transfer protocol.

In Figure 3 we present the batch variant of the OT protocol of [MRR20]. The protocol is essentially the naïve batching of the single-instance protocol, except we use a tagged KA and use different tags for each KA output.

Theorem 9. *When instantiated with a secure batch POPF and a tag-non-malleable KA scheme (Definition 3) with strongly random responses (Definition 4), the OT protocol in Figure 3 is a UC secure batch endemic OT (Figure 2), if the POPF's output satisfies $\mathcal{N} = \mathsf{KA}.\mathcal{M}_2$.*

Proof. Correctness of the POPF and KA clearly show that the protocol is correct in the case where both parties are honest. When both parties are corrupt, the simulator has direct access to both parties and can simulate the real protocol by just running it. This leaves the two interesting cases, where one party is malicious and the other is honest. We prove each case by giving first a simulator, then a sequence of hybrids showing indistinguishability. The hybrids start from the real world and end at the ideal world: the simulator composed with an ideal batch endemic OT.

Simulator for Malicious Sender: The simulator uses $\mathbb{H}_{\mathsf{HSim}}$ instead of $\mathbb{H}$ to implement the local setup. It then waits until the sender provides its protocol message m_S. It creates fresh random values $b_{i,j} \in \mathsf{KA}.\mathcal{R}$ for $i \in [n], j \in \{0,1\}$, then computes the KA messages $m_{i,j} = \mathsf{KA}.\mathsf{msg}_2(b_{i,j}, m_S)$. Then it chooses $\phi_i \leftarrow \mathsf{HSim}(m_{i,0}, m_{i,1})$ and sends $\phi_1, \ldots, \phi_n$ as the simulated protocol message from the honest receiver. Finally, it submits $r_{i,j} = \mathsf{KA}.\mathsf{key}_2(b_{i,j}, m_S, i \parallel j)$ to the ideal functionality, for $i \in [n]$ and $j \in \{0,1\}$ (as the endemic OT values).

Sequence of Hybrids for Malicious Sender: Starting at the real interaction between malicious sender and honest receiver:

1. Replace local setup $\mathbb{H}$ with $\mathbb{H}_{\mathsf{HSim}}$. This change is indistinguishable by the Indistinguishable Local Setups property of the POPF.

2. Change how ϕ_i is generated:

$$\text{replace} \boxed{\begin{array}{l} b_i \leftarrow \mathsf{KA}.\mathcal{R} \\ m_{R,i} = \mathsf{KA}.\mathsf{msg}_2(b_i, m_S) \\ \phi_i \leftarrow \mathsf{Program}(c_i, m_{R,i}) \end{array}} \text{with} \boxed{\begin{array}{l} b_i \leftarrow \mathsf{KA}.\mathcal{R} \\ m_{i,c_i} = \mathsf{KA}.\mathsf{msg}_2(b_i, m_S) \\ m_{i,1-c_i} \leftarrow \mathsf{KA}.\mathcal{M} \\ \phi_i \leftarrow \mathsf{HSim}(m_{i,0}, m_{i,1}) \end{array}}$$

   This is indistinguishable by the Honest Simulation property. Recall that this property requires b_i, m_{i,c_i} to come from a distribution $\mathfrak{D}$ over $\{0,1\}^* \times \mathcal{N}$ where the marginal distribution of the second element is indistinguishable from uniform. This holds because KA has strongly random responses.

3. Change how $m_{i,1-c_i}$ is sampled:

$$\text{replace} \boxed{\begin{array}{l} b_i \leftarrow \mathsf{KA}.\mathcal{R} \\ m_{i,c_i} = \mathsf{KA}.\mathsf{msg}_2(b_i, m_S) \\ m_{i,1-c_i} \leftarrow \mathsf{KA}.\mathcal{M} \\ \phi_i \leftarrow \mathsf{HSim}(m_{i,0}, m_{i,1}) \end{array}} \text{with} \boxed{\begin{array}{l} b_{i,0}, b_{i,1} \leftarrow \mathsf{KA}.\mathcal{R} \\ m_{i,0} = \mathsf{KA}.\mathsf{msg}_2(b_{i,0}, m_S) \\ m_{i,1} = \mathsf{KA}.\mathsf{msg}_2(b_{i,1}, m_S) \\ \phi_i \leftarrow \mathsf{HSim}(m_{i,0}, m_{i,1}) \end{array}}$$

   Later references to b_i become references to b_{i,c_i}. This is indistinguishable because KA has strongly random responses.

This final hybrid describes the ideal world. The receiver's inputs c_i are not used to simulate protocol messages to the sender; they are used only to determine which $r_{i,j} \stackrel{\text{def}}{=} \mathsf{KA}.\mathsf{key}_2(b_{i,j}, m_S)$ the receiver takes as output. In the ideal world the simulator sends identically defined $r_{i,j}$ to the ideal functionality, which uses the receiver's c_i inputs to determine which ones to deliver as the receiver's output.

Simulator for Malicious Receiver: The simulator uses $\mathbb{H}_{\mathsf{Extract}}$ instead of $\mathbb{H}$ to implement the local setup. It generates m_S in the same way as an honest sender, and sends it to the corrupted receiver. When the receiver provides $\phi_1, \ldots, \phi_n$, the simulator runs $c_i = \mathsf{Extract}(\phi_i)$ for all $i \in [n]$, and submits them to the ideal functionality. It also computes $r_{i,c_i} = \mathsf{KA}.\mathsf{key}_1(a, \mathsf{Eval}(\phi_i, c_i), i \parallel c_i)$, and submits these to the ideal functionality as well (as the endemic OT values).

Sequence of Hybrids for Malicious Receiver:

1. Replace local setup $\mathbb{H}$ with $\mathbb{H}_{\mathsf{Extract}}$, an indistinguishable change.

2. Rearrange how $r_{i,j}$ are computed:

$$\text{replace} \boxed{\begin{array}{l}\text{for } j \in \{0,1\}: \\ \quad r_{i,j} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, j), i \parallel j)\end{array}}$$

$$\text{with} \boxed{\begin{array}{l}c_i \leftarrow \mathsf{Extract}(\phi_i) \\ r_{i,c_i} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, c_i), i \parallel c_i) \\ r_{i,1-c_i} = \mathsf{KA.key}_1(a, \mathsf{Eval}(\phi_i, 1-c_i), i \parallel 1-c_i)\end{array}}$$

   This is indistinguishable because running $\mathsf{Extract}$ has no effect on the local setup's internal state.

3. For each $i \in [n]$ and $j \in \{0,1\}$, create an oracle $F_{i,j} = y \mapsto \mathsf{KA.key}_1(a, y, i \parallel j)$. Then rewrite the computation of $r_{i,j}$ in terms of these oracles as $r_{i,j} = F_{i,j}(\mathsf{Eval}(\phi_i, j))$. In Lemma 10 we show that every oracle $F_{i,j}$ is a 1-weak random oracle.

4. Change how $r_{i,1-c_i}$ is chosen:

$$\text{replace} \boxed{\begin{array}{l}c_i \leftarrow \mathsf{Extract}(\phi_i) \\ r_{i,c_i} = F_{i,c_i}(\mathsf{Eval}(\phi_i, c_i)) \\ r_{i,1-c_i} = F_{i,c_i-1}(\mathsf{Eval}(\phi_i, 1-c_i))\end{array}} \text{with} \boxed{\begin{array}{l}c_i \leftarrow \mathsf{Extract}(\phi_i) \\ r_{i,c_i} = F_{i,c_i}(\mathsf{Eval}(\phi_i, c_i)) \\ r_{i,1-c_i} \leftarrow \mathsf{KA}.\mathcal{K}\end{array}}$$

   This change is indistinguishable by the Uncontrollable Outputs property. Since each $F_{i,j}$ is a 1-weak RO, we can apply the Uncontrollable Outputs property once for each i to make the change described here.

This final hybrid describes the ideal world. After seeing the receiver's protocol message, the simulator extracts c_i values and also computes values r_{i,c_i} which will be part of the sender's output. The other OT values in the sender's output ($r_{i,1-c_i}$) are sampled uniformly, just as in the ideal world. □

Lemma 10. *For any tag-non-malleable key agreement KA with strongly random responses, and for any set of tags $\mathcal{T}$, the following distribution outputs a key agreement message and a collection of $|\mathcal{T}|$ weak random oracles from $\mathsf{KA}.\mathcal{M}_2$ to $\mathsf{KA}.\mathcal{K}$.*

$$\boxed{\begin{array}{l}a \leftarrow \mathsf{KA}.\mathcal{R} \\ m_S := \mathsf{KA.msg}_1(a) \\ \text{for } \tau \in \mathcal{T}: \\ \quad F_\tau := x \mapsto \mathsf{KA.key}_1(a, x, \tau) \\ \text{return } m_S, \{F_\tau\}_{\tau \in \mathcal{T}}\end{array}}$$

Proof. We need to show that every F_τ is a weak random oracle. We describe a sequence of hybrids starting from the real weak random oracle distribution and ending at random.

1. Sample the input x and compute y early, when the oracle F_τ is created rather than when the weak RO experiment is run.

2. Instead of sampling $x \leftarrow \mathsf{KA}.\mathcal{M}_2$, sample $b \leftarrow \mathsf{KA}.\mathcal{R}$ and set $x = \mathsf{KA.msg}_2(b, m_S)$. This is indistinguishable by the strongly random responses property of KA.

3. We are now computing $y = \mathsf{KA.key}_1(a, x, \tau)$ for a random KA message x, then giving oracle access to $\mathsf{KA.key}_1(a, x', \tau')$ (from the other oracles $F_{\tau'}$), but only for $\tau' \neq \tau$. This is exactly the same as the real distribution for a tag-non-malleable KA, so it is indistinguishable to switch to the random distribution by randomly sampling $y \leftarrow \mathsf{KA}.\mathcal{K}$ instead.

4. Use strongly random responses again, to sample $x \leftarrow \mathsf{KA}.\mathcal{M}_2$ and remove b.

5. Delay the sampling of x, y until the 1-weak RO distribution is run.

□

Our protocol considers an underlying KA with sequential messages. Yet Diffie-Hellman-based KA protocols have independent messages that can be sent in any order. We call such a KA protocol **1-flow**, where $\mathsf{KA.msg}_2(b)$ is independent of m_S. When the KA is 1-flow, the OT protocol can also be made 1-flow by sending both messages in parallel.

Theorem 11. *Our OT protocol (Figure 3) becomes a 1-flow UC secure batch endemic OT when* KA *is 1-flow.*

Proof. This theorem largely the same as Theorem 9 from the previous one, but with key changes. In the 1-flow instance, the adversary may rush the other party, requiring them to send their message first before responding. For malicious receiver the adversary already went last, but it's different for malicious sender.

When the sender is corrupt, the simulator instead generates $\phi_1, \ldots, \phi_n$ with HSim before receiving m_S, as each of the receiver's messages from the key agreement may now be sampled independently of the sender's. The hybrid proof continues as before, after replacing $\mathsf{KA.msg}_2(b, M_S)$ with $\mathsf{KA.msg}_2(b)$. □

5 New/Improved POPF Constructions

In this section, we describe several suitable POPF constructions for the batch OT protocol.

5.1 Ideal Cipher (EKE)

Our first POPF is inspired by the EKE password-authenticated key exchange protocol of Bellovin & Merritt [BM92]. POPF was created as a generalization of an ideal cipher in the EKE protocol, and it is no surprise that in fact an ideal cipher is a POPF. The full definition is in Figure 4. We are not aware of prior work pointing out the connection between EKE and oblivious transfer. But it is easy to see that an ideal cipher is useful for OT: the adversary can know the trapdoor to at most one of $E^{-1}(0, \phi)$ and $E^{-1}(1, \phi)$.

The local setup $\mathbb{H}$ is simply an ideal cipher. Actually, we have defined $\mathbb{H}$ in a way that is indistinguishable from an ideal cipher — it chooses oracle responses uniformly, instead guaranteeing that each $E(x, \cdot)$ is a permutation. By a standard PRF/PRP switching lemma, the difference is indistinguishable, and this choice makes the description of $\mathbb{H}$ simpler. $\mathbb{H}_{\mathsf{HSim}}$ is similar to $\mathbb{H}$, but it programs E^{-1} so that $\mathsf{Eval}(\phi, i) = r_i$, to satisfy the honest simulation property.

In $\mathbb{H}_{\mathsf{Extract}}$, $\mathsf{Extract}(\phi)$ finds the first ideal cipher call that produced ϕ — either as the input to an E^{-1} query or the output of an E query. The idea is that once ϕ has appeared in some ideal

$\mathcal{M} := \mathcal{N}$

$\underline{\mathsf{Program}(x, y)}$:
  return $E(x, y)$

$\underline{\mathsf{Eval}(\phi, x)}$:
  return $E^{-1}(x, \phi)$

$\mathbb{H}$

$T :=$ empty list
$\underline{E(x, y)}$:
  if $\exists \phi.\, (x, y, \phi) \in T$:
    return ϕ
  $\phi \leftarrow \mathcal{M}$
  append (x, y, ϕ) to T
  return ϕ
$\underline{E^{-1}(x, \phi)}$:
  if $\exists y.\, (x, y, \phi) \in T$:
    return y
  $y \leftarrow \mathcal{N}$
  append (x, y, ϕ) to T
  return y

$\mathbb{H}_{\mathsf{HSim}}$

$T := \{\}$
// E and E^{-1} are same as in $\mathbb{H}$
$\underline{\mathsf{HSim}(r_0, r_1)}$:
  if $\exists x, \phi.\, (x, r_x, \phi) \in T$:
    return $\perp$
  $\phi \leftarrow \mathcal{M}$
  append $(0, r_0, \phi)$ to T
  append $(1, r_1, \phi)$ to T
  return ϕ

$\mathbb{H}_{\mathsf{Extract}}$

$T := \{\}$
// E and E^{-1} are same as in $\mathbb{H}$
$\underline{\mathsf{Extract}(\phi)}$:
  find first $(x^*, y^*, \phi) \in T$:
    return x^*
  if none exist:
    return 0

Figure 4: Batch 2-POPF based on an ideal cipher.

cipher query, future forward queries to E give output ϕ only with negligible probability. Hence, all future calls that involve ϕ must be of the form $E^{-1}(\cdot, \phi)$, meaning that the adversary has no control over the outputs of these queries (which are outputs of Eval). This is precisely the property needed for a POPF.

Theorem 12. *Figure 4 defines a secure and correct batch 2-POPF with all distinguisher advantages except for Uncontrollable Outputs bounded by* $O\left(\frac{q^2}{|\mathcal{N}|}\right)$*, when the adversary makes* q *ideal cipher lookups. Uncontrollable Outputs instead has advantage bounded by* $q\,\mathrm{Adv}(wRO) + O\left(\frac{q^2}{|\mathcal{N}|}\right)$*, where* $\mathrm{Adv}(wRO)$ *is the distinguisher advantage against the 1-weak RO* F*.*

Proof. We have deferred the security proofs for the POPF constructions to the appendix. See Appendix B.1. □

5.2 Even-Mansour POPF

In [MRR20] the authors construct a POPF with a 2-round Feistel cipher. Intuitively, a POPF generalizes an ideal cipher, but is strictly weaker. So while an 8-round Feistel cipher is indifferentiable from an ideal cipher, a 2-round Feistel cipher suffices for a POPF. Similarly, we suggest a POPF

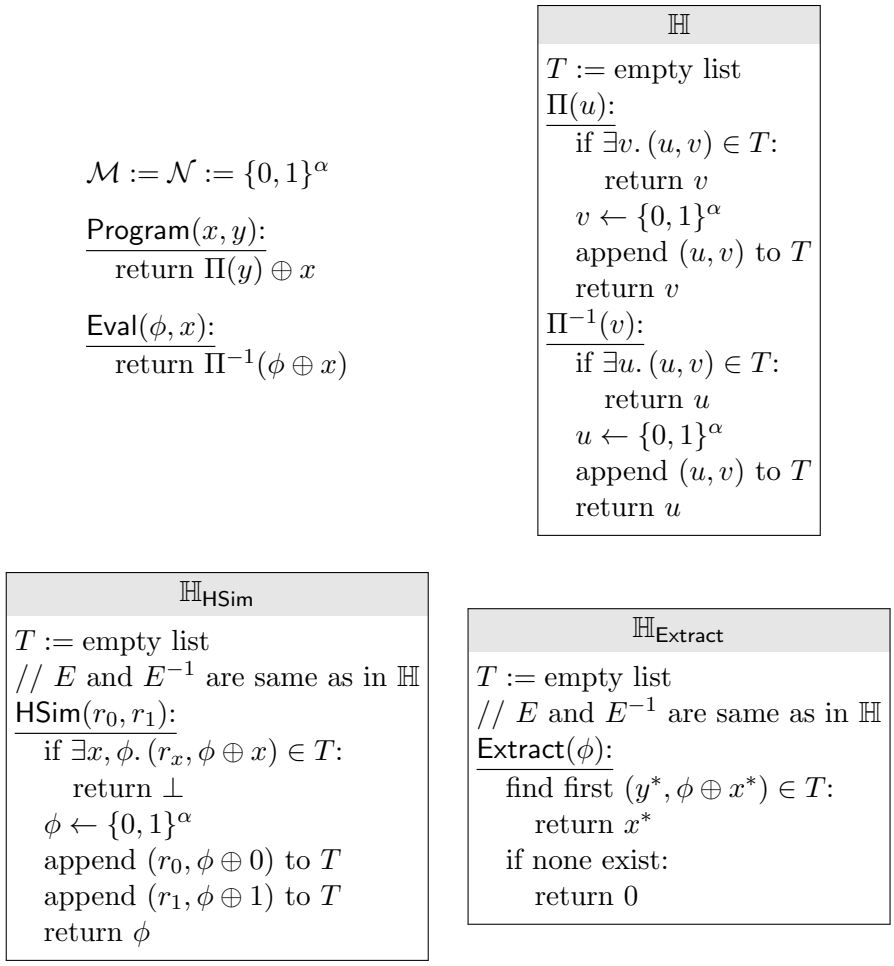

Figure 5: Batch 2-POPF based on an ideal permutation.

based on the Even-Mansour [EM93] construction. While the Even-Mansour construction is not an ideal cipher unless many rounds are added [DSST17], a single round suffices for a POPF.

The construction (Figure 5) is similar to the Ideal Cipher POPF, but with a few changes. The local setup $\mathbb{H}$ is not an ideal cipher, but a simpler ideal random permutation. In the ideal cipher POPF, every query to the oracles included the x-value (as the key of the cipher). In this Even-Mansour POPF the value x is used only by xor'ing with the ideal permutation output — it is not directly available to the simulator (in Extract).

To deal with this challenge, we observe that x can be inferred by the simulator given ϕ. The only situation where x is ambiguous given ϕ is when $\Pi(y_1) \oplus x_1 = \phi = \Pi(y_2) \oplus x_2$ for distinct bits x_1, x_2. This event implies $\Pi(y_1) \oplus \Pi(y_2) = x_1 \oplus x_2 = 1$, which is negligibly likely for forward queries to Π. This turns out to be enough for the simulator to extract. The construction generalizes to strings x which are significantly shorter than the ideal permutation output.

Theorem 13. *Figure 5 defines a secure and correct batch 2-POPF where the distinguisher advantage is $O(q^2 2^{-\alpha})$ when the adversary makes q ideal permutation lookups, except for Uncontrollable Outputs which allows an additional advantage of $q\,\mathrm{Adv}(wRO)$.*

Proof. We have deferred the POPF security proofs to the appendix. See Appendix B.2. □

$\mathcal{M} := \mathbb{G}^2$

$\underline{\mathsf{Program}(x, y)}$:
  $s_{1-x} \leftarrow \mathbb{G}$
  $s_x = y \cdot (H_x(s_{1-x}))^{-1}$
  return (s_0, s_1)

$\underline{\mathsf{Eval}((s_0, s_1), x)}$:
  return $s_x \cdot H_x(s_{1-x})$

$\mathbb{H}$

record calls in a transcript T
$\underline{H_x(u)}$:
  if $\exists v.\, (v \leftarrow H_x(u)) \in T$:
    return v
  $v \leftarrow \mathbb{G}$
  return v

$\mathbb{H}_{\mathsf{HSim}}$

record calls in a transcript T
$U :=$ empty assoc. array
$\underline{H_x(u)}$:
  if $\exists v.\, (v \leftarrow H_x(u)) \in T$:
    return v
  if $U[x, u]$ defined:
    return $U[x, u]$
  $v \leftarrow \mathbb{G}$
  return v
$\underline{\mathsf{HSim}(r_0, r_1)}$:
  $\phi = (s_0, s_1) \leftarrow \mathcal{M}$
  $U[0, s_1] := s_0^{-1} \cdot r_0$
  $U[1, s_0] := s_1^{-1} \cdot r_1$
  return ϕ

$\mathbb{H}_{\mathsf{Extract}}$

record calls in a transcript T
// H_x is the same as in $\mathbb{H}$
$\underline{\mathsf{Extract}((s_0, s_1))}$:
  find first query $H_{x^*}(s_{1-x^*})$ in T:
    return x^*
  if none exist:
    return 0

Figure 6: Batch 2-POPF based on the OT construction of Masny-Rindal [MR19]. Here $H_0, H_1 : \mathbb{G} \to \mathbb{G}$ are random oracles, and $(\mathbb{G}, \cdot)$ is a group.

5.3 Masny-Rindal POPF

This next POPF is inspired by the OT construction of Masny and Rindal [MR19]. Using this POPF in the context of Figure 3 we see that the Masny-Rindal OT protocol for 1-out-of-2 OT[6] is then a specific instance of our protocol. The description of the POPF can be found in Figure 6.

The local setup $\mathbb{H}$ consists of two random oracles H_0, H_1 whose outputs are a group $\mathbb{G}$. In the resulting OT protocol, the KA scheme must have protocol messages that reside in this group. $\mathbb{H}_{\mathsf{HSim}}$ is similar to $\mathbb{H}$, but it also tracks the values r_0, r_1 that have been given to $\mathsf{HSim}(R)$. To satisfy the honest simulation property, it further programs the random oracles H_x to be consistent:

$$\mathsf{Eval}(\phi, x) = s_x \cdot H_x(s_{1-x}) = s_x \cdot (s_x)^{-1} \cdot r_x = r_x.$$

$\mathbb{H}_{\mathsf{Extract}}$ is also very similar to $\mathbb{H}$, but it also tracks chronological order of the oracle queries. $\mathsf{Extract}(\phi)$, upon seeing $\phi = (s_0, s_1)$, checks if s_{1-x^*} was a query to the random oracle H_{x^*}, for either $x^* \in \{0, 1\}$. $\mathsf{Extract}(\phi)$ then chooses the first query (chronologically) and returns the associated x^*, or chooses x^* arbitrarily to be 0 if neither call was made. As in the original proof in [MR19] the main idea is that for the adversary to program ϕ, they need to query on one of the two s_x values in

[6] Generalizing to 1-out-of-N for polynomial N works the same as in [MR19].

$\mathcal{N} := \mathbb{G}$
$\mathcal{M} := \mathbb{G} \times \mathbb{F}$

$\underline{\mathsf{Program}(x, y)}$:
  $u \leftarrow \mathbb{F}$
  $t := H(x, u)^{-1} \cdot y$
  $s := u - \iota(t)x$
  return s, t

$\underline{\mathsf{Eval}((s, t), x)}$:
  return $H(x, \iota(t)x + s) \cdot t$

$\mathbb{H}$

record calls in a transcript T
$\underline{H(x, u)}$:
  if $\exists v.\, (v \leftarrow H(x, u)) \in T$:
    return v
  $v \leftarrow \mathbb{G}$
  return v

$\mathbb{H}_{\mathsf{HSim}}$

record calls in a transcript T
$U :=$ empty assoc. array
$\underline{H(x, u)}$:
  if $\exists v.\, (v \leftarrow H(x, u)) \in T$:
    return v
  if $U[x, u]$ defined:
    return $U[x, u]$
  $v \leftarrow \mathbb{G}$
  return v
$\underline{\mathsf{HSim}(r_0, r_1)}$:
  $(s, t) \leftarrow \mathbb{G} \times \mathbb{F}$
  $U[0, \iota(t)\,0 + s] := r_0 \cdot t^{-1}$
  $U[1, \iota(t)\,1 + s] := r_1 \cdot t^{-1}$
  return (s, t)

$\mathbb{H}_{\mathsf{Extract}}$

record calls in a transcript T
// H is the same as in $\mathbb{H}$
$\underline{\mathsf{Extract}((s, t))}$:
  find first query $H(x^*, \iota(t)x^* + s)$:
    return x^*
  if none exist:
    return 0

Figure 7: Variant of the Feistel POPF in [MRR20], where one random oracle has been replaced with multiplication in a finite field $\mathbb{F}$. ι is an injection with an efficient left inverse ι^{-1}, *i.e.*, $\forall t.\, \iota^{-1}(\iota(t)) = t$.

order to find the other, unless the "other" is sampled independently, in which case the adversary fails to program.

Theorem 14. *Figure 6 defines a secure and correct batch 2-POPF with all distinguisher advantages except for Uncontrollable Outputs bounded by* $O\left(\frac{q^2}{|\mathbb{G}|}\right)$ *when the adversary makes* q *queries to the random oracles. Uncontrollable Outputs instead has advantage bounded by* $\frac{q^2-q+2}{2}\,\mathrm{Adv}(wRO) + O\left(\frac{q^2}{|\mathbb{G}|}\right)$.

Proof. We have deferred this proof to Appendix B.3. □

5.4 Streamlined Feistel POPF

[MRR20] propose a POPF based on 2-round Feistel, in which the ϕ value is 3κ bits longer than the underlying value from $\mathcal{N}$. We present an alternative construction (Figure 7) that improves on this

when $\mathbb{G} = \mathcal{N}$ can be represented with less than 3κ bits. This is useful because elliptic curve points usually can be represented with 2κ bits.

As with MRR20, we need $\mathcal{N}$ to be a group $\mathbb{G}$, and the local setup $\mathbb{H}$ is a hash function H mapping into $\mathbb{G}$. However, instead of a second random oracle $H'(x, T)$, we use an injection ι from $\mathbb{G}$ into a finite field $\mathbb{F}$. The hash call $H'(x, T)$ in one of the Feistel rounds is then replaced with multiplication $\iota(T)x$. ι is required to have an efficiently computable left inverse ι^{-1}.

These changes eliminate the main bad event in the security proof of MRR20, which occurs when the adversary manages to delay making the H' query, which the simulator needs to see in order to find what T the adversary chose, until after the simulator needs to use T to program H. The simulator can now find T directly using ι^{-1}.

Theorem 15. *The streamlined Feistel POPF in Figure 7 is a secure and correct batch 2-POPF. The distinguisher advantage is* $O\left(\frac{q^2}{|\mathbb{G}|}\right)$ *when the adversary makes* q *ideal permutation lookups, except for Uncontrollable Outputs which allows an additional advantage of* $\frac{q^2-q+2}{2}\,\mathrm{Adv}(wRO)$.

Proof. We have deferred this proof to Appendix B.4. □

The original 2-round Feistel POPF in [MRR20] also satisfies our new definitions. We omit the proof because it is substantially similar to the proof of Theorem 15, just preserving a few more ideas from [MRR20].

6 Suitable Key Agreement Choices

Our batched OT protocol requires a tagged KA in which the receiver's protocol messages are indistinguishable from the uniform distribution over the domain of the POPF (outputs of Eval). In this section we discuss several choices for KA, including one not considered in [MRR20] but which is well-suited to the batch setting.

The main challenge is that traditional DHKA on an elliptic curve is not enough. Under the usual encoding (the x-coordinate), points on the curve are easily distinguishable from random strings, while it is more natural to define a POPF operating on strings. Hence, some care is involved in making the POPF and KA compatible.

6.1 Curve Mappings

In [MRR20], the authors suggest two ways to achieve compatibility between POPF and KA.

One choice is to ensure that the KA protocol messages are uniform *bit strings*. This can be done using the Elligator technique of [BHKL13] to encode curve elements. Elligator is an injective and efficiently invertible function ι from $\{0,1\}^\kappa$ to a large subset of the elliptic curve. If some party wishes to make their KA protocol message a uniform string, they simply sample from points in the image of ι. This is achieved in practice by re-sampling a DH exponent until the resulting curve point is in $\iota(\{0,1\}^\kappa)$. If the range of ι is a large fraction of the elliptic curve, then the expected number of re-samples is small. See Figure 8 for a formal description of tagged Elligator ECDHKA.

Another choice is to ensure that the POPF Eval function only outputs values on the curve. In the POPF construction of [MRR20] this can be achieved by instantiating a random oracle that gives outputs in the curve.

Both of these techniques incur nontrivial computational overhead. The elligator approach requires resampling each curve element some constant number of times on average. The state of the art techniques for hashing-to-curve [BCI+10, FFS+10, TK17] have cost roughly 25% that of an exponentiation on the curve, and the POPF requires at least 2 hash-to-curve operations per party.

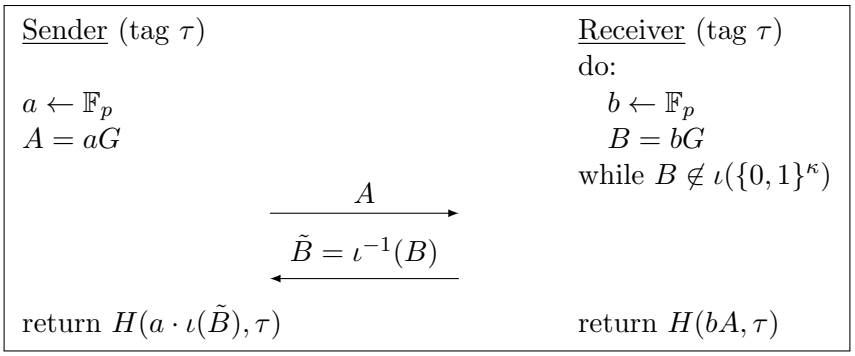

Figure 8: Tagged Elligator ECDHKA. G is a generator of the curve and ι is the injective Elligator mapping of [BHKL13].

6.2 Möller Variant of ECDHKA

We now suggest a more efficient approach that is well suited for the batch setting. Before continuing, let us give a brief review of elliptic curves. For the remainder of this section, we will consider curves over prime fields with order larger than 3. Further results and descriptions can be found in Silverman [Sil09].

Definition 16. *An **elliptic curve** $E_{a,b}$ over a field $\mathbb{F}_p$ is defined by a congruence of the form $Y^2 = X^3 + aX + b$ parameterized by elements $a, b \in F_p$ such that $4a^3 + 27b^2 \neq 0$. The elements of $E_{a,b}$ are given by tuples (X, Y) satisfying the congruence along with a neutral element $\mathcal{O}$, the **point at infinity**.*

We may equip this set with a group law called the chord-and-tangent law such that we arrive at a commutative group where the usual Diffie-Hellman problems are believed to be hard.

Definition 17. *Given an elliptic curve $E_{a,b}$ over a field $\mathbb{F}_p$ and $c \in \mathbb{F}_p$, we may consider the elliptic curve $E'_c : cY^2 = X^3 + aX + b$. If c is a quadratic residue in $\mathbb{F}_p$ then E' is isomorphic to E, otherwise, E' is called the (quadratic) **twist** of E.*

As a twist of a given curve is unique up to isomorphism, we may consider, singly, a primary curve and its twist curve. It follows from the definition that any $x \in \mathbb{F}_p$ is the abscissa (x-coordinate) of a point on E or of a point on the twist E'.

Lemma 18. *Let $c \neq 0$ be a quadratic non-residue in the field $\mathbb{F}_p$, and let $E_{a,b}$ be an elliptic curve over $\mathbb{F}_p$ with twist E'_c. Then for every $x \in \mathbb{F}_p$:*

1. *If $x^3 + ax + b$ is a non-zero qudratic residue, then $(x, \pm\sqrt{x^3 + ax + b})$ are points on $E_{a,b}$. Furthermore, $(x^3 + ax + b)/c$ is a quadratic non-residue and x is not the abscissa of any point on E'_c*
2. *If $x^3 + ax + b$ is a quadratic non-residue, then x is not a point on $E_{a,b}$. Furthermore, $(x^3 + ax + b)/c$ is a quadratic residue and $(x, \pm\sqrt{(x^3 + ax + b)/c})$ are points on E'_c.*
3. *If $x^3 + ax + b = 0$, then $(x, 0)$ is a point on $E_{a,b}$ and E'_c.*

This idea is of importance as for many curves and applications, only the abscissa of a point is needed. This means that we can work with bitstrings using the implicit mapping defined above.

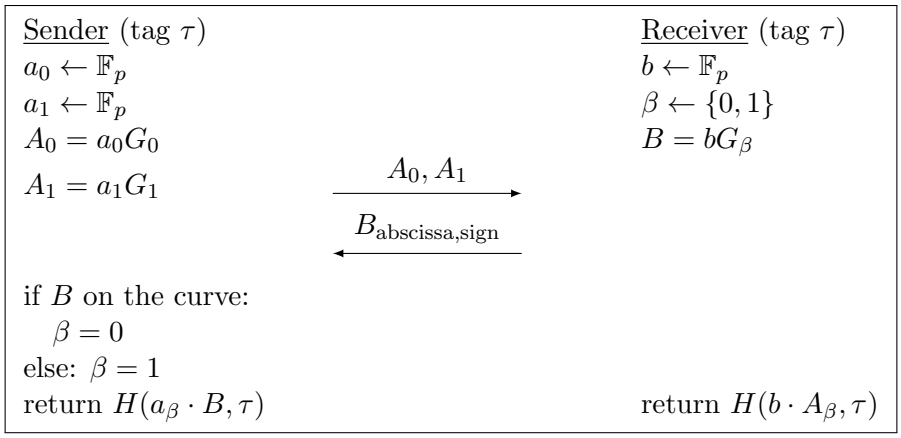

Figure 9: Möller tagged ECDHKA. G_0 is a generator of the curve and G_1 is a generator of its twist.

Furthermore, there are a similar number of points on the twist as there are on the curve. If one were to toss a coin $b \leftarrow \{0,1\}$, and then sample an x-coordinate of a random curve point (if $b = 0$) or a random twist point (if $b = 1$), the result would be statistically close to the uniform distribution on the set of bitstrings.

Lemma 19 ([CFGP06, Corollary 11])**.** *Given a curve $E_{a,b}$ and its twist E'_c over $\mathbb{F}_p$, where $2^q - p < 2^{q/2}$ (i.e., p is very close to a power of 2), the following distribution is indistinguishable from the uniform distribution in $\{0,1\}^q$*

$$\mathcal{D} = \{\beta \leftarrow \{0,1\}, x_0 \leftarrow [E_{a,b}]_{abscissa}, x_1 \leftarrow [E'_c]_{abscissa} : K = x_\beta\},$$

with statistical distance

$$\delta = \frac{1}{2} \sum_{x \in \mathbb{F}_p} \left| \Pr_{K \leftarrow \mathbb{F}_{2^q}}[K = x] - \Pr_{K \leftarrow \mathcal{D}}[K = x] \right| \leq \frac{1 + \sqrt{2}}{2^{q/2}}.$$

This suggests the key agreement approach in Figure 9. The receiver will sample an x-coordinate as above. The sender cannot anticipate the receiver's choice, so she prepares a DH message on both the curve and the twist, then chooses the correct one to compute the final key. Lemma 19 establishes that the receiver's KA message is statistically indistinguishable from the uniform distribution on strings.

Note that the sender sends two curve/twist elements instead just one as in standard DHKA. However, in batched OT it is exactly this sender message that is reused across all OT instances. Hence a slight increase in its size has minimal effect on the overall OT protocol's efficiency.

Similar approaches to representation have been used used in the context of PAKE [BMN01], pseudo-random permutations [Kal91], authenticated key exchange [CFGP06], and by Möller [Möl04] in the context of ElGamal.

6.3 Curve Choice and Security

We now discuss the security of the Möller variant (tagged) KA protocol. The choice of curve must satisfy the following

- The finite field must have order at least $2^q - 2^{q/2}$.

- The curve and its twist must be cryptographically secure.
- The curve and its twist must be cyclic.

More specifically, we need a security property similar to the **oracle Diffie-Hellman (ODH)** assumption [ABR01]. That definition is as follows:

Definition 20 ([ABR01])**.** *Let $\mathbb{G}$ be a cyclic group of order n, with generator g, and let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. Then the **oracle Diffie-Hellman (ODH)** assumption holds in $\mathbb{G}$ with respect to H if the following distributions are indistinguishable, for all $\mathcal{A}$ that do not query their oracle at g^b.*

$a, b \leftarrow [n]$ def $\mathcal{H}_a(X) = H(X^a)$ $K = H(g^{ab})$ return $\mathcal{A}^{\mathcal{H}_a}(g^a, g^b, K)$	$a, b \leftarrow [n]$ def $\mathcal{H}_a(X) = H(X^a)$ $K \leftarrow \{0,1\}^\ell$ return $\mathcal{A}^{\mathcal{H}_a}(g^a, g^b, K)$

Our applications require a variant of ODH where the hash function H takes an additional *tag* argument:

Definition 21. *Let $\mathbb{G}$ be a cyclic group of order n, with generator g, and let $H : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^\ell$ be a hash function. Then the **tagged oracle Diffie-Hellman (TODH)** assumption holds in $\mathbb{G}$ with respect to H if the following distributions are indistinguishable, for all tags τ^* and all $\mathcal{A}$ that do not query their oracle with second argument τ^*:*

$a, b \leftarrow [n]$ def $\mathcal{H}_a(X, \tau) = H(X^a, \tau)$ $K = H(g^{ab}, \tau^*)$ return $\mathcal{A}^{\mathcal{H}_a}(g^a, g^b, K)$	$a, b \leftarrow [n]$ def $\mathcal{H}_a(X, \tau) = H(X^a, \tau)$ $K \leftarrow \{0,1\}^\ell$ return $\mathcal{A}^{\mathcal{H}_a}(g^a, g^b, K)$

In [ABR01] the authors show that standard ODH is secure in the generic group model when H is a random oracle. This proof is easily adapted to the new TODH assumption as well.

Proposition 22. *Möller tagged DHKA (Figure 9) satisfies tag nonmalleability (Definition 3) if the TODH assumption holds in both the curve and its twist.*

A further small optimization is possible for Montgomery curves. The exponentiation algorithm only depends on the x-coordinate of its input and is uniform for both the curve and its twist, in the sense that the usual exponentiation algorithm for the curve also correctly exponentiates in the twist if the input is on the twist. So if the sender in Figure 9 chooses $a_0 = a_1$ then there is no need to check whether the receiver's B is on the curve or twist. Instead, the sender simply exponentiates B without any checking. However, security of this optimization requires that a kind of TODH assumption hold for the curve and twist jointly (instead of separately/independently for the curve and for the twist).

6.3.1 Instantiation

When creating a concrete instantiation of Möller ECDHKA, we chose to use Curve25519 [Ber06]. The main reasons for this choice were:

1. The base field $\mathbb{F}_p$ is of prime order $2^{255} - 19 > 2^{255} - 2^{255/2}$.

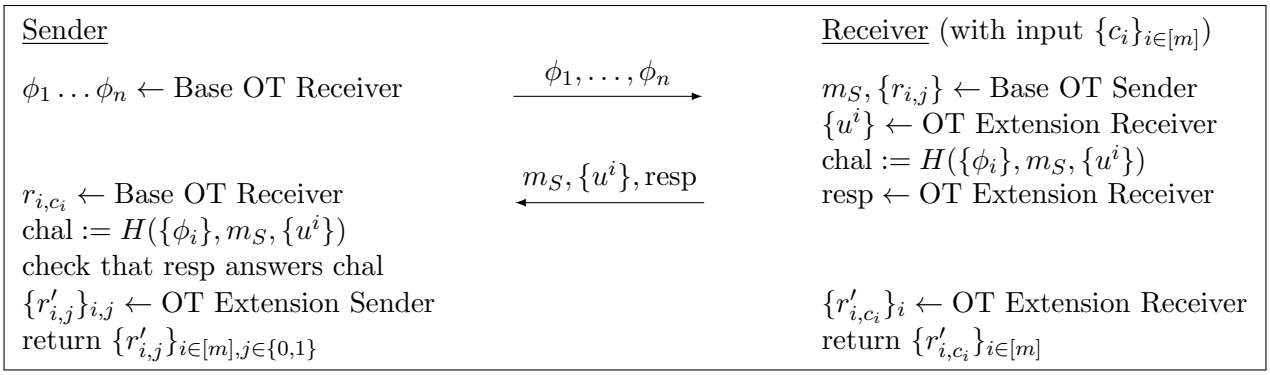

Figure 10: Sketch of the composition of our batch OT protocol with the KOS OT extension protocol, in 2 rounds.

2. Curve25519 is explicitly designed to have a twist that is as secure as the curve itself.
3. Curve25519 can take full advantage of Montgomery Ladders for scalar multiplication which allows us to use only the abscissa in computations.
4. Curve25519 and its twist have large prime subgroups of size $\#E/8$ and $\#E'_c/4$.

Curve25519 also provides additional evidence for the security of the above optimization of setting $a_0 = a_1$, because [Ber06] recommends not checking whether a given point is on the curve or twist before performing scalar multiplication. This optimization is why Curve25519 was chosen to have a secure twist, and in fact the reference implementation does not check if an elliptic curve point is on the curve. This requires a similar additional security assumption to our optimization because it uses the same key for both the curve and its twist.

7 2-round Endemic OT Extension

When our protocol is used for base OTs, we can achieve a 2-round Endemic OT extension protocol, if the Fiat-Shamir heuristic is used. First, recall that our batch OT protocol is 1-flow when instantiated with a 1-flow KA protocol, *e.g.*, any Diffie-Hellman-based KA protocol. This gives us the flexibility to send base OT messages in any order.

Second, we summarize the 1-out-of-2 OT extension protocol of [KOS15]:

- The parties perform base OTs
- The receiver (who is base OT sender) sends data as in all IKNP-based [IKNP03] extension protocols.
- To protect against a malicious receiver, the sender gives a random challenge
- The receiver sends a response to this challenge, which the sender checks.

We can order the messages of the base OTs so that the receiver can send their IKNP data along with their base OT sender message. Additionally, we can collapse the malicious consistency check using the Fiat-Shamir heuristic, since the sender's challenge is random. The resulting OT extension protocol is sketched in Figure 10.

In related work, [CSW20] show how to use the Chou-Orlandi base OT protocol to achieve 3-round OT extension. This is inevitable since their base OTs already require 3 rounds. [BCG+19]

show a 2-round OT extension protocol based on newer "silent OT" techniques. Note however that both these papers achieve chosen message OT, while Figure 10 only achieves endemic OT and would require a third round to derandomize the sender's messages.

8 Performance Evaluation

In this section, we will explore the concrete performance benchmarks of multiple instantiations of the protocol in Figure 3.

8.1 Implementation Details

We implemented[7] our protocol inside the `libote` OT extension library [Rin], modifying the library to use Rijndael-256 [DR99, BÖS11] as a standin for an ideal cipher and `libsodium` [Den20] to implement elliptic curve operations. The library uses Blake2 [ANWW13] as a standin for a random oracle. We then tested the protocols on a machine running on an Intel Xeon E5-2699 v3 CPU, without assembly optimizations or multi-threading. For benchmarking, each protocol was run in a batch of 128 OTs for two settings of simulated latency and bandwidth limiting. The two settings are meant to shed light on the LAN vs WAN environments that these protocols may run in. The number of OTs to run was chosen to provide a realistic setting in the case of 128 base OTs as is common in OT extension.

We compared the following implementations:

- Chou-Orlandi (Simplest OT).
- Naor-Pinkas OT
- Masny-Rindal (Endemic OT), with and without reusing the sender's message. This protocol uses hash-to-curve operations.
- Our protocol instantiated with Möller's DHKA and various POPFs. We used the original Feistel POPF of MRR, as well as the POPFs presented in Section 5. Because the messages from Möller's scheme are uniformly random bit strings, our these POPFs avoid the hash-to-curve operations that are needed in [MR19]. We did not evaluate the Even-Mansour POPF (Figure 5), since its performance would be identical to the EKE POPF (Figure 4) when both the ideal cipher and ideal permutation are instantiated with Rijndael.
- Our protocol with traditional DHKA, and all POPF instantiations excluding EKE and Masny-Rindal. We did not implement the EKE POPF using DHKA; however this could be accomplished using Elligator or a similar mapping to construct an ideal cipher on a subset of the curve points. We did not implement our protocol with Masny-Rindal POPF as it would be nearly identical to the Masny-Rindal protocol.

8.2 Results & Discussion

The performance benchmarks can be found in Table 2 for both settings.

As we would expect, when comparing the three instances of Masny-Rindal OT, each with their own improvement, we see a marked increase in efficiency. Specifically, reusing the sender's message reduced the total time spent by both parties by 18% / 11% in the low latency and high bandwidth

[7] Source code is at `https://github.com/Oreko/popfot-implementation`.

Protocol	Security	Sender (ms)	Receiver (ms)
0.1ms latency, 10000Mbps bandwidth cap			
Simplest OT [CO15] (Sender-reuse)	standalone	35	17
Naor-Pinkas OT [NP01] (Sender-reuse)	standalone	43	34
Endemic OT [MR19] (No reuse)	UC	79	42
Endemic OT (Sender-reuse)	UC	62	37
Ours (Feistel POPF [MRR20] – DHKA)	UC	82	40
Ours (Streamlined Feistel POPF Figure 7 – DHKA)	UC	80	40
Ours (Feistel POPF — Möller DHKA)	UC	49	26
Ours (Streamlined Feistel POPF — Möller DHKA)	UC	50	27
Ours (Masny-Rindal POPF Figure 6 — Möller DHKA)	UC	48	27
Ours (EKE POPF Figure 4 — Möller DHKA)	UC	50	25
30ms latency, 100Mbps bandwidth cap			
Simplest OT [CO15] (Sender-reuse)	standalone	105	111
Naor-Pinkas OT [NP01] (Sender-reuse)	standalone	101	107
Endemic OT [MR19] (No reuse)	UC	161	53
Endemic OT (Sender-reuse)	UC	137	53
Ours (Feistel POPF [MRR20] – DHKA)	UC	155	47
Ours (Streamlined Feistel POPF Figure 7 – DHKA)	UC	155	47
Ours (Feistel POPF — Möller DHKA)	UC	128	44
Ours (Streamlined Feistel POPF — Möller DHKA)	UC	128	44
Ours (Masny-Rindal POPF Figure 6 — Möller DHKA)	UC	128	44
Ours (EKE POPF Figure 4 — Möller DHKA)	UC	128	44

Table 2: Running time of batch OT protocols, for batch size of 128.

setting / the high latency and low bandwidth setting, respectively. Moving to Möller's KA caused an additional 24% / 9% improvement, respectively, for the Masny-Rindal construction. On average, for the three protocols with both DHKA and Möller DHKA versions (Masny-Rindal and the two Feistel POPF variants) we saw an improvement of 32% / 13%, respectively, when moving to Möller's KA.

As expected, the Simplest OT protocol outperforms our instantiations for the sender since it uses fewer exponentiations in the group. One point to take note of in the evaluation data is the large gap in the performance for the receiver between the Naor-Pinkas and Simplest / Blazing OT constructions and the POPF and Masny-Rindal constructions in the high latency / low bandwidth setting. This is due to the different flow requirements between the two sets of protocols. Simplest OT and Naor-Pinkas constructions all require an additional flow (or two) which, in the WAN setting, will acrue more time for the party which needs to wait. It then follows that the advantages of our protocol over Simplest OT is our UC security and round/flow complexity.

References

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.

[ANWW13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: Simpler, smaller, fast as MD5. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 119–135. Springer, Heidelberg, June 2013.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 967–980. ACM Press, November 2013.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

[BMN01] Colin Boyd, Paul Montague, and Khanh Quoc Nguyen. Elliptic curve based password authenticated key exchange protocols. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 487–501. Springer, Heidelberg, July 2001.

[BÖS11] Joppe W. Bos, Onur Özen, and Martijn Stam. Efficient hashing using the AES instruction set. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 507–522. Springer, Heidelberg, September / October 2011.

[CFGP06] Olivier Chevassut, Pierre-Alain Fouque, Pierrick Gaudry, and David Pointcheval. The Twist-AUgmented technique for key exchange. In Moti Yung, Yevgeniy Dodis, Aggelos

Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 410–426. Springer, Heidelberg, April 2006.

[CMR] Brent Carmer, Alex J. Malozemoff, and Marc Rosen. swanky: a suite of rust libraries for secure multi-party computation. https://github.com/GaloisInc/swanky.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, Heidelberg, August 2015.

[CSW20] Ran Canetti, Pratik Sarkar, and Xiao Wang. Blazing fast OT for three-round UC OT extension. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 299–327. Springer, Heidelberg, May 2020.

[Den20] Frank Denis. The sodium cryptography library, Nov 2020.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[DSST17] Yuanxi Dai, Yannick Seurin, John P. Steinberger, and Aishwarya Thiruvengadam. Indifferentiability of iterated Even-Mansour ciphers with non-idealized key-schedules: Five rounds are necessary and sufficient. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 524–555. Springer, Heidelberg, August 2017.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, November 1993.

[FFS+10] Reza R. Farashahi, Pierre-Alain Fouque, Igor E. Shparlinski, Mehdi Tibouchi, and J. Felipe Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. Cryptology ePrint Archive, Report 2010/539, 2010. http://eprint.iacr.org/2010/539.

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

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 8–26. Springer, Heidelberg, August 1990.

[Kal91] Burton S. Kaliski Jr. One-way permutations on elliptic curves. *Journal of Cryptology*, 3(3):187–199, January 1991.

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. https://eprint.iacr.org/2020/521.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

[Möl04] Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS 2004*, volume 3193 of *LNCS*, pages 335–351. Springer, Heidelberg, September 2004.

[MR19] Daniel Masny and Peter Rindal. Endemic oblivious transfer. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 309–326. ACM Press, November 2019.

[MRR20] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 425–442. ACM Press, November 2020.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 381–396. Springer, Heidelberg, February 2017.

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

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[Sma] Nigel Smart. SCALE-MAMBA: Secure computation algorithms from LEuven, multiparty algorithms basic argot. https://homes.esat.kuleuven.be/~nsmart/SCALE/.

[TK17] Mehdi Tibouchi and Taechan Kim. Improved elliptic curve hashing and point representation. Designs, Codes and Cryptography, 2017.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

A Correlation Attack on Naïvely Batched MRR-OT

Consider the following strategy for a corrupt receiver, against naïve batching of the MRR OT protocol.

- The sender first sends its (reused) KA message $A = g^a$.

- In the first instance, run honestly with choice bit $c_1 = 0$. Generate $\phi_1 = \mathsf{Program}(0, g^b)$ for known b.
- The receiver can compute the value $\tilde{B} = \mathsf{Eval}(\phi_1, 1)$. The security of the POPF is that the receiver has no control over this value (*i.e.*, doesn't know its discrete log). The sender will compute OT output from this instance $r_{1,1} = \tilde{B}^a$.
- In the second instance, set $\phi_2 = \mathsf{Program}(0, \tilde{B} \cdot g^s)$ for known s. This means that the sender will compute OT output from this instance

$$r_{2,0} = \mathsf{Eval}(\phi_2, 0)^a = (\tilde{B} g^s)^a = \tilde{B}^s g^{as} = r_{1,1} g^{as} = r_{1,1} A^s$$

Note that the receiver can indeed compute A^s, and therefore it knows the ratio $r_{2,0}/r_{1,1}$.

Note also that if s is uniform then $\tilde{B} \cdot g^s$ is distributed uniformly. From the simulator's point of view, the receiver's behavior is *identically distributed* to honest behavior — running $\mathsf{Program}(0, X)$ for a uniform group element X. Hence, even if the ideal functionality is weakened to allow the receiver to specify correlations among the OT values, in this protocol the simulator has no way of detecting which correlation is appropriate.

B Security Proofs for POPFs

B.1 Security Proof for Ideal Cipher (EKE) POPF

First, we have that $\mathbb{H}_{\mathsf{HSim}}, \mathbb{H}, \mathbb{H}_{\mathsf{Extract}}$ are indistinguishable, because they are all identical other than HSim and $\mathsf{Extract}$. $\mathsf{Extract}$ only reads $\mathbb{H}_{\mathsf{Extract}}$'s state, but does not modify it. Correctness follows directly from the correctness property of ideal ciphers.

There is an invariant that must be maintained for the ideal cipher to continue working properly. For any x, ϕ there must not be $y_1 \neq y_2$ such that $(x, y_1, \phi) \in T$ and $(x, y_2, \phi) \in T$. Similarly, for any x, y there must not be $\phi_1 \neq \phi_2$ such that $(x, y, \phi_1) \in T$ and $(x, y, \phi_2) \in T$. If either case happened, the output of E or E^{-1} would not be well-defined. A birthday bound shows that E and E^{-1} maintain this invariant. HSim explicitly aborts if it is asked to break the invariant.

Honest Simulation: Recall that the y^* sampled by $\mathfrak{D}$ must be uniform. By a birthday bound it is unique, not overlapping any previous y in T, with all but negligible probability. Then when REAL_PHI samples $\phi \leftarrow \mathsf{Program}(x^*, y^*)$, E will choose a uniformly random ϕ, the same distribution as sampled by HSim. It will also add (x^*, y^*, ϕ) to T. A similar birthday bound allows us to assume that this ϕ is also unique. Then a $E^{-1}(1-x^*, \phi)$ query will be freshly random, so we can equivalently sample it ahead of time in a random value r_{1-x^*} and add $(1-x^*, r_{1-x^*}, \phi)$ to T. But this is exactly what happens in SIM_PHI, as the abort in HSim will not occur because r_0 and r_1 will be unique.

Uncontrollable Outputs: We provide a sequence of hybrids, starting from the real distribution and ending at the ideal distribution.

1. Create an empty associative array Z at the start of $\mathbb{H}_{\mathsf{Extract}}$. Inside E^{-1}, whenever y is sampled as freshly random, compute and save $Z[y] = F(y)$. Then use the precomputed value as r in Uncontrollable Outputs instead of finding it again; since $y = \mathsf{Eval}(\phi, 1-x^*)$ is calculated $Z[y]$ must have been precomputed. This step just rearranges the order of computations, and so is indistinguishable.

2. For the first E^{-1} query, instead of finding $F(y)$, sample $Z[y]$ as a uniformly random value in $\mathcal{O}$. We would like to use the 1-weak RO's security to prove that this is indistinguishable, but it seems like we are using F multiple times, and so cannot use the security property. However, this multiple use is illusory, as only single value in Z will ever be used and the rest will be discarded.

   More concretely, if we construct a reduction from this change in the hybrid proof to the 1-weak RO security, without loss of generality we can assume that the adversary aborts if this first $Z[y]$ does not end up being used to find r. If it is not used then the two distributions are identical anyway. Now the other computations of F are completely unused and can be removed, allowing us to reduce to 1-weak RO security.

3. Repeat Step 2 for subsequent E^{-1} queries.

4. Undo the changes in step 1. That is, delay randomly sampling the entries of Z until Uncontrollable Outputs is run, so r will be uniformly random.

The advantage is bounded by adding up the advantages of every step. All security properties used a constant number of birthday bounds, which give the adversary an advantage of $O\left(\frac{q^2}{N}\right)$. Additionally, Uncontrollable Outputs used the security of the 1-weak RO once in each E^{-1} query, for a total advantage of $q\,\mathrm{Adv}(wRO)$.

B.2 Security Proof for Even-Mansour POPF

The proof is very similar to that of Theorem 12, and we will only describe the differences. The invariant no longer mentions x, and just says that each u should have at most one v and vice versa. Honest Simulation works similarly to before, pre-programming the randomness that Eval will produce. It will preserve the invariant as long as it does not produce the same ϕ or $\phi \oplus 1$ as one produced previously — ignoring a single bit should not affect collision resistance.

For extraction, the only additional complication is arguing that the x^* produced by $\mathsf{Extract}$ is the only one where $\mathsf{Eval}(\phi, x^*)$ is not freshly random on its first call. The only way for it to not be random is for there to have been a previous $\Pi(u)$ call that returned $\phi \oplus x$, but this cannot happen for multiple x as we have assumed that the other bits of ϕ are unique. $\mathsf{Extract}$ checks T to find the unique call $v = \Pi(u)$ where this happened, which must have been the first, then finds x^* such that $v = \phi \oplus x^*$.

B.3 Security Proof for Masny-Rindal POPF

First, we have $\mathbb{H}_{\mathsf{HSim}} \approx \mathbb{H} \approx \mathbb{H}_{\mathsf{Extract}}$, because they are all identical when neither HSim nor $\mathsf{Extract}$ have been called, though some have extra bookkeeping. $\mathsf{Extract}$ only reads $\mathbb{H}_{\mathsf{Extract}}$'s state, but does not modify it. Correctness is ensured as if $\phi = (s_0, y \cdot (H_1(s_0))^{-1}) \leftarrow \mathsf{Program}(1, y)$ then $y = y \cdot (H_1(s_0))^{-1} \cdot H_1(s_0) = \mathsf{Eval}(\phi, 1)$, and similarly for $x^* = 0$.

Honest Simulation: In the generation of $\phi = (s_0, s_1)$, we have that s_{1-x^*} is uniform by construction, and furthermore since y^* is sampled from $\mathfrak{D}$ and must be uniform we have that $\mathsf{Program}$ will generate uniform s_{x^*}. s_{x^*} is distinguishable only if $H_{x^*}(s_{1-x^*})$ had previously been queried which for uniform s_{1-x^*} only happens with probability at most $\frac{q}{|\mathbb{G}|}$.

We also have uniqueness of s_0, s_1 with all but negligible probability by an application of a birthday bound over $\mathbb{G}$, so using U to program H_x will not interfere with any previous H_x queries or other HSim calls. Finally, we have that $\mathsf{Eval}(\phi, 1 - x^*) = s_{1-x^*} \cdot H_{1-x^*}(s_{x^*})$ is close to uniform

(since the query $H_{1-x^*}(s_{x^*})$ will be fresh with probability $1-\frac{q}{|\mathbb{G}|}$), so we can sample it early and save it in U. This is exactly how HSim functions in the ideal world. Finally, by the enforced consistency of HSim we have that the outputs r_0, r_1 from Eval are the values provided to HSim. This can be verified as if $\phi = (s_0, s_1) \leftarrow \mathsf{HSim}(r_0, r_1)$ then $\mathsf{Eval}(\phi, x) = s_x \cdot H_x(s_{1-x}) = s_x \cdot (s_x)^{-1} \cdot r_x = r_x$.

Uncontrollable Outputs: Similarly to the proof of Claim 4.3 in [MR19], we would like to guess a query $H_{x^*}(u^*)$. Following MRR20, we will call this query an anchor query. The idea is that this is the query made by $\mathsf{Program}$, or however the adversary constructed ϕ. Any subsequent query $H_{1-x^*}(u)$ can be programmed to be $u^{*-1} \cdot y$ to make $\mathsf{Eval}(\phi, 1-x^*) = y$ if we guessed correctly. We will know we guessed correctly if later u^* is part of the ϕ that is input to $\mathsf{Extract}$.

However, instead of guessing a query like in [MR19], we will use a hybrid proof to get the same result. Some hybrids will make changes that are only useful if a guess is correct, but do nothing if the guess is wrong. Here is our sequence of hybrids starting with an interaction with the real protocol and ending with the ideal world.

1. Create a new associative array Z at the start of $\mathbb{H}_{\mathsf{Extract}}$. When a uniformly random value v is sampled in $H_x(u)$, look for possible anchor queries by iterating over all previous queries $H_{1-x}(u^*)$, and in each iteration, compute and save $Z[x, u^*, u] = F(u^* \cdot v)$.

2. Use the precomputed value $Z[1-x^*, s_{1-x^*}, s_{x^*}]$ as r in the Uncontrollable Outputs distribution, instead of finding it again with $F(\mathsf{Eval}(\phi, 1-x^*))$, if it has already been computed.

3. For $1 \le i \le q$ and $1 \le j < i$, repeat the following sequence of hybrids. That is, perform these transformations for the ith query to H and jth iteration of the loop over prior queries in H, for a total of $\frac{q(q-1)}{2}$ repetitions.

   (a) Instead of sampling $v \leftarrow \mathbb{G}$ in the ith query $H_x(u)$, use u^* from the jth iteration of the loop over possible anchor queries to sample $y \leftarrow \mathbb{G}$ and set $v = u^{*-1} \cdot y$.

   (b) In the jth iteration of the ith query to H, instead of computing $Z[\phi, x] = F(y)$, sample $Z[\phi, x]$ as a uniformly random value in $\mathcal{O}$. This change is indistinguishable because F is a 1-weak RO.[8]

   (c) Undo the changes in step 3a, so v is sampled as $v \leftarrow \mathbb{G}$ again.

4. Undo the changes in step 1. That is, wait until the Uncontrollable Outputs distribution is run before sampling the entries in Z. If $Z[1-x^*, s_{1-x^*}, s_{x^*}]$ is present, the Uncontrollable Outputs distribution now gets a uniformly random r instead of the output of F.

5. Finally, also replace r with random if it does not appear in Z. In this case, either $H_0(s_1)$ or $H_1(s_0)$ must not have been queried before, as otherwise whichever was queried first would be the anchor query and then in the second query r would be precomputed and saved in Z. Either x^* will be the query that was made, or neither were queried — either way, $H_{x^*}(s_{1-x^*})$ must not have been queried. Therefore $\mathsf{Eval}(\phi, 1-x^*)$ must return a fresh uniformly random value, and the 1-weak RO property allows us to replace F's output with random.

For Honest Simulation, the adversary just gets a birthday bound advantage $O\left(\frac{q^2}{N}\right)$. But in Uncontrollable Outputs we use the security of 1-weak RO, which allows them an additional advantage. We use it $\frac{q(q-1)}{2}$ times in step 3b, and one additional time in step 5. Therefore, Uncontrollable Outputs allows the adversary an advantage of $\frac{q^2-q+2}{2}\,\mathrm{Adv}(wRO)$, on top of the birthday bounds.

[8] Although it appears that F is used in multiple places, only a single one is actually used in the end. See the proof for the EKE POPF for details.

B.4 Security Proof for Feistel POPF

All three $\mathbb{H}$s are clearly indistinguishable on their common interface, as without any HSim queries they behave identically. For correctness, notice that

$$\begin{aligned}\mathsf{Eval}(\mathsf{Program}(x^*, y^*), x^*) &= H(x^*, \iota(t)x^* + u - \iota(t)x^*) \cdot t \\ &= H(x^*, u) \cdot H(x^*, u)^{-1} \cdot y = y\end{aligned}$$

Honest Simulation: REAL_PHI chooses u uniformly randomly, so the $H(x^*, u)$ query will return fresh randomness as it has negligible probability of overlapping with any other query. Therefore $\phi = (s, t)$ will be uniformly random, the same distribution as produced by HSim. Next, we just need to prove that HSim successfully programs Eval, *i.e.*, that r_0 and r_1 will match between REAL_PHI and SIM_PHIs. It succeeds if the second if-statement in $H(x, \iota(t)x + s)$ is triggered, because then $\mathsf{Eval}(\phi, x)$ will produce $H(x, \iota(t)x + s) \cdot t = r_x \cdot t^{-1} \cdot t = r_x$.

There are two ways that it could fail: either $H(x, \iota(t)x + s)$ had already been queried before HSim was called, or $U[x, \iota(t)x + s]$ gets overwritten by another HSim query. The former case has negligible probability because there are at most q previous queries $H(x, u)$, each would cause a failure only if $s = u - \iota(t)x$, and s is chosen uniformly at random after the $H(x, u)$ query. For the latter case, notice that every $\phi = (s, t)$ defines a unique line $u = \iota(t)x + s$. A pair of such lines would have to intersect at some $x \in \{0, 1\}$ in order for U to be overwritten. They can only intersect for a single value of x, and since both lines are uniformly random, this x will be uniformly random in $\mathbb{F}$, so there is negligible probability of an intersection for $x \in \{0, 1\}$.[9]

Uncontrollable Outputs: We use MRR20's notion of anchor queries for this proof. An anchor query is a query made during $\mathsf{Program}$ that can be used by $\mathbb{H}_{\mathsf{Extract}}$ to identify ϕ before it is revealed by the adversary. More specifically, a query $H(x^*, u^*)$ is the anchor query if it is the first query on the line $u^* = \iota(t)x^* + s$. It is, in fact, the query that $\mathsf{Extract}$ searches for in order to find x^*. The anchor query is needed in order to find t early and program the subsequent H queries such that Eval outputs a random value for the weak random oracle.

MRR20 guessed the anchor query, taking a factor q security loss, and we will do something similar with hybrids. In a chain of hybrids we guess a possible anchor query and make some changes that make progress if the guess was correct and do nothing if we are wrong. Once the anchor query $H(x^*, u^*)$ has been made, on each subsequent query $H(x, u)$ we assume it is on the same $u = \iota(t)x + s$ line and use this to find ϕ and program $H(x, u)$. Specifically, $t = \iota^{-1}\left(\frac{u-u^*}{x-x^*}\right)$ can be found from the slope of the line through the points $(x^*, u^*), (x, u)$, and $s = u - \iota(t)x$ is the u-axis intercept. If this assumption is wrong there is no harm, similarly to the anchor query.

We use the following sequence of hybrids, from the real distribution to the ideal distribution.

1. Create an empty associative array Z at the start of $\mathbb{H}_{\mathsf{Extract}}$. Inside H, whenever v is sampled as freshly random, iterate over all previous queries $H(x^*, u^*)$ for $x^* \neq x$ to look for possible anchor queries. For each such query, compute $\phi = (s, t)$ as described above. Skip to the next H-query if this ϕ came up in a previous iteration, as then it is impossible for $H(x^*, u^*)$ to be the anchor query for ϕ. Compute and save $Z[\phi, x] = F(v \cdot t)$.

2. Use the precomputed value $Z[\phi, 1 - x^*]$ as r in the Uncontrollable Outputs distribution if it is present, instead of computing it again as $F(\mathsf{Eval}(\phi, 1 - x^*))$.

[9] When handling exponentially large x this becomes problematic, but can be fixed by hashing x with another random oracle H' before multiplying by ι, so that the adversary would need to solve a hard preimage problem to find the x corresponding to an intersection.

3. For $1 \leq i \leq q$ and $1 \leq j < i$, repeat the following sequence of hybrids. That is, perform these transformations for the ith query to H and jth possible anchor query, for a total of at most $\frac{q(q-1)}{2}$ repetitions.

   (a) Instead of sampling $v \leftarrow \mathbb{G}$ in the ith query $H(x, u)$, use the inferred $\phi = (s, t)$ from the jth possible anchor query to sample $y \leftarrow \mathbb{G}$ and set $v = y \cdot t^{-1}$.

   (b) In the jth iteration in the ith query to H, instead of computing $Z[\phi, x] = F(y)$, sample $Z[\phi, x]$ as a uniformly random value in $\mathcal{O}$. This change is indistinguishable because F is a 1-weak RO.[10]

   (c) Undo the changes in step 3a, so v is sampled as $v \leftarrow \mathbb{G}$ again.

4. Undo the changes in step 1. That is, delay randomly sampling the entries in Z until Uncontrollable Outputs is run. If $Z[\phi, 1 - x^*]$ is present, the Uncontrollable Outputs distribution now gets a uniformly random r instead of the output of F.

5. Finally, if $Z[\phi, 1 - x^*]$ is not present, replace the output r of the 1-weak RO with random. In this case $H\big(1 - x^*, \iota(t)(1 - x^*) + s\big)$ cannot have been queried before, as otherwise either the anchor query would be at $1 - x^*$ not x^*, or the anchor query $H(x^*, \iota(t)x^* + s)$ would have been made before the other query and so $Z[\phi, 1 - x^*]$ would be present, which are both contradictions. Therefore, the call to $\mathsf{Eval}(\phi, 1 - x^*)$ in the Uncontrollable Outputs distribution must return fresh randomness, and then the 1-weak RO property allows us to replace r with random.

Again, we bound the advantage by summing the advantages of each step in the hybrid proof. Excluding the birthday bounds, the only advantage the adversary gets is in Uncontrollable Outputs, when we use the 1-weak RO property. We use it $\frac{q(q-1)}{2}$ times in step 3b, once for each pair of oracle queries, because we have to loop over every previous query as a possible anchor query. Finally, we use it one last time in step 5. Therefore, Uncontrollable Outputs allows the adversary an advantage of $\frac{q^2-q+2}{2} \mathrm{Adv}(wRO)$, on top of the birthday bounds.

[10] Although it may seem that F is used multiple times, only one of these values will actually be used in the end. For more details, see the proof for the EKE POPF.