On randomness complexity of 1-private protocols*

Samuel Dittmer† and Rafail Ostrovsky‡

Abstract

In the field of information-theoretic cryptography, randomness complexity is a key metric for protocols for private computation, that is, the number of random bits needed to realize the protocol. Although some general bounds are known, even for the relatively simple example of 1-private computation of n-party AND, the exact complexity is unknown.

We study two settings. First, we consider the model of Goyal, Ishai, and Song (Crypto '22) where helper parties without any inputs are allowed to assist in the computation. In this setting, we show that *two* random bits always suffice to compute an arbitrary Boolean circuit C 1-privately: a single designated inputless helper flips the two bits and privately distributes the derived one-time bits to the other helper parties and the input parties as they are needed. We give an explicit construction using seven helper parties per AND gate and three helper parties per XOR gate (plus the single global randomness dealer). Moreover, two random bits are necessary already for the AND functionality (by a reduction to the standard no-helper model together with the lower bound of Kushilevitz, Ostrovsky, Prouff, Rosén, Thillard and Vergnaud (TCC '19), and therefore the worst-case helper-party randomness complexity is exactly 2 bits.

Second, in the setting without helper parties, we improve the upper bound from Couteau and Rosén (Asiacrypt '22) on the (asymptotic) randomness complexity of n-party AND from 6 to 5 bits. That is, we give a 1-private protocol for computing the AND of n parties' inputs requiring 5 bits of randomness, for all $n \geq 6$. Our construction, like that of Couteau and Rosén, uses a single party to flip the 5 bits and distribute the required derived values during the execution. Our approach to both problems is built around a more systematic exploration of techniques for *recycling randomness* across sub-computations.

As part of resolving the second problem, we isolate an exact local-independence combinatorial object called a *Sliding-Window Independence Generator*, or a **SWIG**. A (k, m)-SWIG is a linear generator from a k-bit seed to $m \geq k$ output bits, where every cyclic length-k sliding window chosen from m output bits is perfectly uniform. We give an explicit (k, m)-SWIG for every $k \geq 1$ and every $m \geq k$ and use a $(5, n-1)$-SWIG in our no-helper AND protocol.

*Preliminary version appeared in ICALP 2026.

†Stealth Software Technologies, Inc. Email: *sam@stealthsoftwareinc.com*

‡UCLA, Email: *rafail@cs.ucla.edu*

1 Introduction

We consider a concrete question in information-theoretic cryptography: How many bits of randomness are required for n parties to compute a Boolean circuit C on their inputs in a 1-private protocol with semi-honest security?

We are motivated by one practical and one theoretical topic of interest in computer science. On the practical side, reducing randomness complexity can improve the usability of cryptographic protocols. In real world cryptography, random bits must either be collected from physical sources or generated from a pseudo-random number generator applied to some initial seed. There are costs and security risks associated with both approaches, see e.g. [IKL+13, ISW03] and the discussion in [GIS22].

On the theoretical side, there is a noteworthy connection between the complexity of 1-private computation and circuit complexity. As Kushilevitz, Ostrovsky, and Rosén showed [KOR96], any Boolean function f that requires $\omega(1)$ bits of randomness to securely compute requires a circuit of size $\omega(n)$. Therefore, 1-private randomness complexity of Boolean functions implies circuit lower bounds. Extensions to t-privacy were considered in [CKOR97, CKOR00]. We study two protocol settings and isolate one combinatorial object that drives the second construction:

- **Helper-party model:** We allow additional parties that do not have any inputs to assist in the computation [GIS22]. The idea is that one helper party flips a small, truly random seed (here, only two global random bits suffice) and privately distributes the derived one-time bits to the remaining helpers and input parties. The helper parties are then used as "one-time gadgets" for gates. A key point is that each gadget is used only once, so it suffices that *each individual gadget transcript* is statistically (in fact, perfectly) independent of the true secrets when at most one party is corrupted.

- **No-helper model (AND only):** We remove all helper parties and instead focus on n-party AND where each player has an input bit [KOR96, KOP+19, KOP+21, CR22]. Again, we want to reuse a tiny seed, but now the helper roles must be *emulated* by parties with inputs. Therefore, each party will play several roles in the protocol. That is, every party participates in multiple steps and receives multiple messages across the computation. Therefore, we must assemble masks so that the entire multi-step view of any single party is consistent with fresh masks in each one-party view, even though everything is globally derived from a single 5-bit seed.

- **Sliding-Window Independence Generator:** We isolate a combinatorial object that is behind our no-helper protocol. We define a (k, m)-SWIG as a linear Sliding-Window Independence Generator with a k-bit seed that outputs $m \geq k$ bits for any $m \geq k$. The SWIG construction must guarantee that every consecutive k-bit window of the output, with wrap-around, is uniform. Our AND protocol (without helpers) for n parties uses $(5, n-1)$-SWIG.

We stress that all our definitions and constructions are information-theoretic and do not rely on any (cryptographic or other) assumptions.

1.1 Our Contributions

Let C be a Boolean circuit over the basis $\{\oplus, \wedge\}$ with constants, with n input wires, $k_\oplus$ XOR gates, and $k_\wedge$ AND gates, after local preprocessing. We allow unbounded fanout. Public constants are wires rather than gates. The symbol k in SWIG statements is unrelated to the gate counts $k_\oplus$ and $k_\wedge$.

Theorem 1.1. *The function computed by C admits a perfectly 1-private semi-honest protocol in the helper-party model that uses at most 2 random bits. For the preprocessed circuit, the protocol uses exactly $1 + 3k_\oplus + 7k_\wedge$ inputless helper parties, including the single randomness dealer. Moreover, for $n \geq 3$, the n-party AND functionality requires at least 2 random bits even with arbitrarily many inputless helpers. Hence, for every input length $n \geq 3$, the worst-case randomness complexity in the helper-party model is exactly 2.*

Theorem 1.1 uses only $O(|C|)$ helper parties. Combining this with the protocol-to-circuit direction of Kushilevitz, Ostrovsky, and Rosén [KOR96, Lemma 3] gives the following helper-party analog of their circuit/protocol equivalence. In the reverse direction, helper parties are counted as additional parties when applying the KOR protocol-to-circuit lemma.

Corollary 1.2. *For a Boolean function family $\{f_n : \{0,1\}^n \to \{0,1\}\}$, the family has Boolean circuits of size $O(n)$ if and only if it has perfectly 1-private semi-honest helper-party protocols using at most 2 random bits and $O(n)$ inputless helpers.*

Our second result is the following:

Theorem 1.3. *For every $n \geq 6$, there is a perfectly 1-private semi-honest protocol with a single 5-bit random seed (and no helper parties) for n-party AND of their individual inputs.*

We stress that we do not claim optimality in the no-helper 5 random-seed case; this is an upper bound only. Our no-helper construction is based on an exact combinatorial property that the reused masks must satisfy, which we believe is of independent interest:

Definition 1.4 ((k,m)-SWIG)**.** *For all $k, m \in \mathbb{Z}^+$ such that $m \geq k$ we define a (k,m)-SWIG to be a locally-computable linear map:*

$$G : \mathbb{F}_2^k \to \mathbb{F}_2^m$$

satisfying the following condition: Let $S \leftarrow \mathbb{F}_2^k$ be uniform, and write $G(S) = (Z_0, Z_1, \ldots, Z_{m-1})$ such that for every $j \in \mathbb{Z}_m$, the cyclic window $(Z_j, Z_{j+1}, \ldots, Z_{j+k-1})$ is uniform over $\mathbb{F}_2^k$ (all indices are taken modulo m). Equivalently, for every $j \in \mathbb{Z}_m$ and every $\alpha \in \mathbb{F}_2^k$,

$$\Pr_S[(Z_j, Z_{j+1}, \ldots, Z_{j+k-1}) = \alpha] = 2^{-k}$$

We can also write G by giving explicit vectors $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{m-1} \in \mathbb{F}_2^k$ such that

$$G(S)_i = \langle \mathbf{u}_i, S \rangle$$

for every output coordinate i.

When the output length m is fixed, as a shorthand, we call (k, m)-SWIG a k-SWIG. Thus, a 5-SWIG is a five-bit seed generator that outputs a string of length m from a 5-bit seed such that every length-five sliding window with wraparound in this m-bit 5-SWIG output is uniform.

Theorem 1.5 (5-SWIG of every output length)**.** *For every $m \geq 5$, there is a $(5, m)$-SWIG over $\mathbb{F}_2$.*

In particular, for every $n \geq 6$, there is a $(5, n-1)$-SWIG over $\mathbb{F}_2$. In fact, we show the following stronger version, although it is not used in our AND protocol:

Theorem 1.6 (k-SWIG)**.** *For every $k \geq 1$, $m \geq k$, there is an explicit construction of (k, m)-SWIG over $\mathbb{F}_2$.*

Our closed-form binary construction of (k, m)-SWIGs for every $m \geq k$ can be viewed as an explicit construction of the cyclic-window, fixed-basis specialization of the classical consecutive-basis interpolation theorem of Linial and Tarsi [LT89], and binary n-regular matrix framework of Etzion and Lempel [EL86].

Relation to prior sample-space constructions. Schulman developed a small sample space local neighborhood construction [Sch92]. Since cyclic k-windows, has neighborhood size k and coordinate degree $\Delta = k$, Shulman's construction requires seed length $\lceil k + \log_2 k \rceil$. Another related work is that of Kushilevitz and Mansour [KM97]. In their work, define $D_{m,k}$ be the maximum number of distinct nonzero cyclic-window parity tests for a fixed coordinate. KM sparse-family construction gives seed length $\lceil \log_2(D_{m,k} + 1) \rceil$. Moreoever, once $m \geq 2k-1$, the $D_{m,k} \leq (k+1)2^{k-2}$, becomes equality . Thus, for $k = 5$, KM construction gives 5 bits for $m = 5, 6$ and 6 bits for every $m \geq 7$.

Construction / measure	general k	$k = 5$
Schulman seed length	$\lceil k + \log_2 k \rceil$	8 bits
Schulman support size	$2^{\lceil k+\log_2 k \rceil}$	256
KM sparse seed length	$\lceil \log_2(D_{m,k} + 1) \rceil$	5 bits ($m = 5, 6$) 6 bits ($m \geq 7$)
KM sparse support size	$2^{\lceil \log_2(D_{m,k}+1) \rceil}$	32 ($m = 5, 6$) 64 ($m \geq 7$)
k-SWIG seed length	k	5 bits
k-SWIG support size	2^k	32

SWIG construction optimality for sample spaces. Define a hypergraph H on $\mathbb{Z}_m$ where all cyclic intervals $\{i, i+1, \ldots, i+k-1\}$ for $i \in \mathbb{Z}_m$ are its hyperedges. Observe that in this formulation, k-SWIG on m bit output is an H-wise independent sample space. Furthermore, k-SWIG guarantees that every hyperedge of H is uniform on $\{0,1\}^k$ [HL19]. Equivalently, the $2^k \times m$ matrix of SWIG sliding windows is a cyclic-consecutive test array in which every cyclic block of k columns contains each k-bit word exactly once [GKM11, SWL16]. Furthermore, our construction is optimal since any such sample space or test array must have

at least 2^k rows, as any length-k block must already have 2^k possible binary patterns. Our SWIG construction achieves this lower bound with 2^k rows. Thus, for binary sliding windows, SWIG gives an explicit optimal construction rather than non-constructive (covering array) bounds of [RMS18, MRS19].

Our SWIG construction is surprisingly simple. We define G that takes k-bit seed S, and outputs m bits as follows. Let $m = qk + s$, where $q \geq 1$ and $0 \leq s < k$. We first define vectors $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{m-1} \in \mathbb{F}_2^k$ as q repetitions of the standard basis vectors $e_0, \ldots, e_{k-1}$, followed by $\mathbf{b}_0, \ldots, \mathbf{b}_{s-1}$, (empty if $s = 0$), where N is the least power of two with $N \geq k$, and for $0 \leq t < s$,

$$\mathbf{b}_t = \left(\binom{N-k}{t}, \binom{N-k+1}{t}, \ldots, \binom{N-1}{t} \right) \pmod 2$$

where the i'th output of G is $G(S)_i = \langle \mathbf{u}_i, S \rangle$ for each $0 \leq i \leq m-1$.

1.2 Preliminaries

Bits are elements of $\mathbb{F}_2$, where $a \oplus b = a + b$ and $a \wedge b = ab$. We use the standard simulation definition for perfect 1-privacy for semi-honest players. For a protocol Π, and input vector $\mathbf{a}$, and a party i, the view of party i maintains the identities of the senders and receivers to party i and the chronological order of messages to and from party i. The view of player i in protocol Π is defined as:

$$\mathsf{View}_i^{\Pi}(\mathbf{a}) = (\text{input } a_i \text{ (if any)},\ i\text{'s random tape},\ \text{all } \Pi \text{ messages to/from player } i)$$

In our helper-party construction, one distinguished helper party acts as the randomness dealer. It samples two random bits and sends deterministic functions of them over private point-to-point channels.

Definition 1.7 (Perfect 1-privacy)**.** *A protocol Π that computes a deterministic functionality f with one public output bit is* perfectly 1-private *if the following condition holds. For every party i there is a simulator Sim_i. For every input vector $\mathbf{a}$, the real view $\mathsf{View}_i^{\Pi}(\mathbf{a})$ has exactly the same distribution as $\mathsf{Sim}_i(a_i, f(\mathbf{a}))$ when i is an input party. It has exactly the same distribution as $\mathsf{Sim}_i(f(\mathbf{a}))$ when i is an inputless helper party. The public output broadcast is included in every view.*

The *randomness complexity* of a protocol counts only fresh random bits, and does not count any bits that are derived from (a combination) of these bits. We use cyclic indexing for SWIG outputs: when J is an ordered cyclic set of size m, x_{i+t} is obtained by advancing t positions in J. A cyclic length-k window is the ordered tuple $(x_i, x_{i+1}, \ldots, x_{i+k-1})$. The SWIG condition requires that cyclic windows of length k be uniform. We stress that the SWIG condition does not require ordinary k-wise independence among arbitrary subsets of

positions. Since a SWIG is linear and has a k-bit seed, Definition 1.4 has a useful equivalent form. For each j, the sliding window map

$$S \mapsto (Z_j, Z_{j+1}, \ldots, Z_{j+k-1})$$

is a linear bijection from $\mathbb{F}_2^k$ to $\mathbb{F}_2^k$. That is, our SWIG is specified by public vectors $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{m-1} \in \mathbb{F}_2^k$ and outputs $Z_i = \langle \mathbf{u}_i, S \rangle$ The following simple rank condition is the one we use in our SWIG construction.

Lemma 1.8 (SWIG rank condition)**.** *Public vectors $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{m-1}$ is a (k, m)-SWIG if and only if every length k sliding window, $\mathbf{u}_j, \mathbf{u}_{j+1}, \ldots, \mathbf{u}_{j+k-1}$, has full rank.*

Proof. For any starting position j, the SWIG map is linear, and for a random seed S, the output window is uniformly exactly when this linear map is onto. Since both the domain and codomain have dimension k, being onto is the same as having full rank of k. □

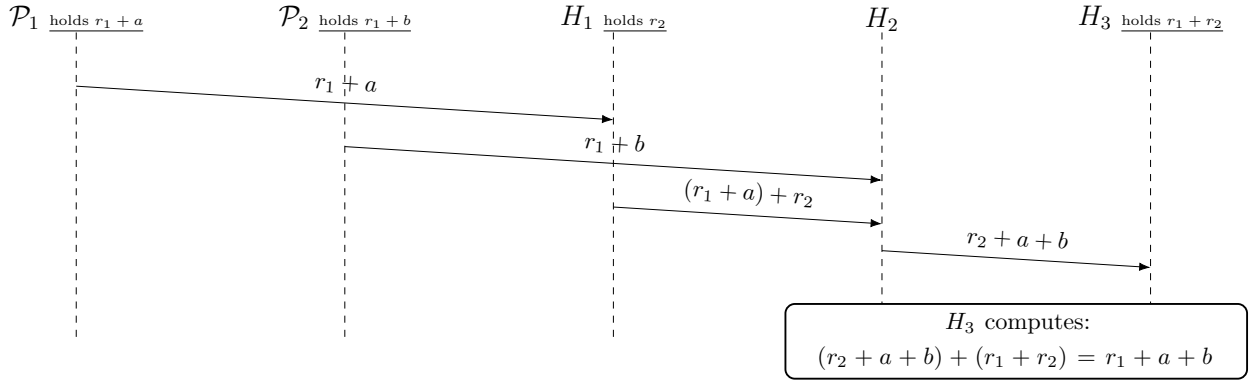
Remark 1.1 (On privacy proofs in this paper). Many gadgets below output masked values rather than public outputs. For such gadgets we prove that for any one corrupted participant, the full local view is simulatable from the participant's allowed input and the public output. The proofs all follow the same blueprint: we describe the local view and rewrite each message as a deterministic expression in uniform masks. We then use the fact that adding a fixed secret bit to a uniform mask leaves a uniform bit. These are role-level statements. In the helper-party circuit protocol, wire holders may later send deterministic copies or public toggles of the same masked wire. Lemma 3.1 explains why appending these public functions preserves perfect simulation. In the no-helper protocol, a real party can play several roles. The privacy proof, therefore, gives a joint simulator for its entire view.

2 Base Protocols

This section provides constant-party gadgets. Throughout the circuit composition, an intermediate value w is stored as $w + r_1$ under a global mask bit r_1. In the helper-party evaluation of Section 3, the input parties know r_1. Intermediate wire holders are helpers and do not know this bit.

2.1 Masked Addition with matching masks

Protocol Interaction Diagram: Masked Addition with three Helpers Π_3



For both $\mathcal{P}_1$ and $\mathcal{P}_2$, observe that we mask their current input bits by the same random r_1. Therefore, if the two masked values were added directly, then $(r_1 + a) + (r_1 + b) = a + b$ and masks would cancel each other. This is not useful for our purposes since we need to keep intermediate values hidden. Therefore, our gadget first temporarily hides the sum under r_2, then uses $r_1 + r_2$ to convert the result back into the masked value under r_1. Thus, H_2 sees only a freshly masked intermediate value.

Masked Addition with three Helpers $\Pi_3(\mathcal{P}_1, \mathcal{P}_2, (H_i)_{i=1}^3, a, b, r_1, r_2)$

Input. $\mathcal{P}_1$ holds $r_1 + a$ and $\mathcal{P}_2$ holds $r_1 + b$. Helpers: H_1 holds r_2, H_2 has no initial input, and H_3 holds $r_1 + r_2$.

Output. H_3 holds $r_1 + a + b$.

Protocol.

1. $\mathcal{P}_1$ sends $r_1 + a$ to H_1.
2. $\mathcal{P}_2$ sends $r_1 + b$ to H_2.
3. H_1 sends $(r_1 + a) + r_2$ to H_2.
4. H_2 sends $(r_1 + b) + ((r_1 + a) + r_2) = r_2 + a + b$ to H_3.
5. H_3 outputs $(r_2 + a + b) + (r_1 + r_2) = r_1 + a + b$.

Figure 1: Π_3: Secure delegated addition when both inputs share the same mask

Lemma 2.1. *For every single corrupted role in the role experiment of Figure 1, the corrupted party's view is distributed independently of (a, b).*

Proof. We check the roles of all five players separately.

The role $\mathcal{P}_1$. Its view and its message are $r_1 + a$, which is independent of a since r_1 is uniform. It is also independent of b. The same argument applies to $\mathcal{P}_2$, whose only visible bit is $r_1 + b$.

The role H_1. The view is $(r_2, r_1+a, (r_1+a)+r_2)$. The first two coordinates are independent by the same argument as for $\mathcal{P}_1$, and the third is a deterministic function. Thus, the distribution is independent of (a, b).

The role H_2. The two input messages are $r_1 + b$ and $(r_1 + a) + r_2$ and are independent of (a, b) for the same reason as above. The outgoing message is their XOR.

The role H_3. H_3 gets as input $r_1 + r_2$ and $r_2 + a + b$. This pair of inputs is an invertible affine image of (r_1, r_2) for every fixed (a, b). The output is a deterministic function of the pair.

Therefore, the simulator for each role can sample the input-independent distributions and append the deterministic outgoing messages. □

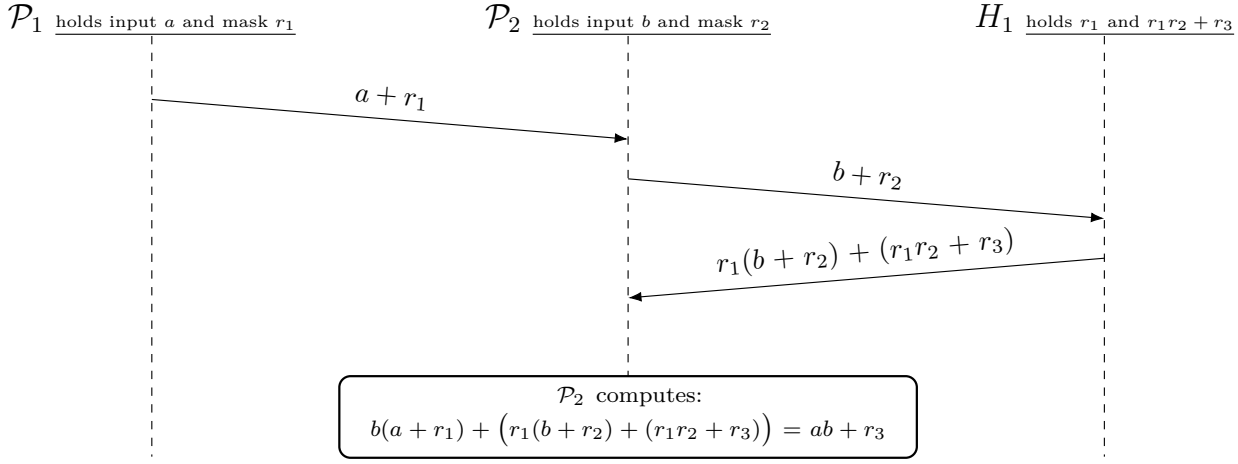
2.2 Masked Multiplication (one helper, three random bits)

The one-helper multiplication gadget uses three random bits r_1, r_2, r_3. Party $\mathcal{P}_1$ sends $a + r_1$. Party $\mathcal{P}_2$ multiplies it by its input b, which results in $ab + br_1$. The helper computes the correction from $b + r_2$, r_1, and $r_1r_2 + r_3$, since

$$r_1(b + r_2) + (r_1r_2 + r_3) = br_1 + r_3$$

$\mathcal{P}_2$ adds the correction to $b(a + r_1)$, which results in $ab + r_3$.

Protocol Interaction Diagram: Masked Multiplication with one helper Π_1



Lemma 2.2 (Privacy for Π_1)**.** *The protocol in Figure 2 is correct and has the following role-level privacy. In the masked-holder variant, the first role sees only $M = a + r_1$ and its outgoing copy. Its view is independent of (a, b). In the input-holder variant, the first role's view is simulatable from a and is independent of b. The second role's view is simulatable from b and is independent of a. This remains true even though it outputs the masked value $ab + r_3$, whose joint distribution with the rest of the second role's view remains simulatable from b. The helper role's view is independent of (a, b).*

Masked Multiplication with one Helper $\mathbf{\Pi}_1(\mathcal{P}_1, \mathcal{P}_2, H_1, a, b, r_1, r_2, r_3)$

Input. The first player role receives the masked value $M = a + r_1$ (or, in the input-holder variant, knows (a, r_1) and computes M). $\mathcal{P}_2$ holds b and r_2. H_1 holds r_1 and $r_1 r_2 + r_3$.

Output. $\mathcal{P}_2$ holds $ab + r_3$.

Protocol.

1. $\mathcal{P}_1 \to \mathcal{P}_2$: send $M = a + r_1$.
2. $\mathcal{P}_2 \to H_1$: send $b + r_2$.
3. H_1 sends $r_1(b + r_2) + (r_1 r_2 + r_3) = br_1 + r_3$ to $\mathcal{P}_2$.
4. $\mathcal{P}_2$ outputs $b(a + r_1) + (br_1 + r_3) = ab + r_3$.

Figure 2: Π_1: Secure delegated multiplication with one helper party

Proof. Correctness follows from

$$b(a + r_1) + \Big(r_1(b + r_2) + (r_1 r_2 + r_3)\Big) = ab + r_3$$

In the masked-holder variant, the first player $\mathcal{P}_1$ gets $M = a + r_1$, which is independent of a. In the input-holder variant, $\mathcal{P}_1$ has (a, r_1) and sends $a + r_1$, where a is his own input.

The second role player knows (b, r_2). It receives $a + r_1$, sends $b + r_2$, receives $br_1 + r_3$, and outputs $ab + r_3$. Conditioned on (b, r_2), the pair $(a + r_1, br_1 + r_3)$ is an affine image of the uniform pair (r_1, r_3). The linear part has matrix $\begin{pmatrix} 1 & 0 \\ b & 1 \end{pmatrix}$ in the variables (r_1, r_3) and is invertible. Hence, the pair is independent of a. The output is a deterministic function of the same local data, so it is included in the simulated view.

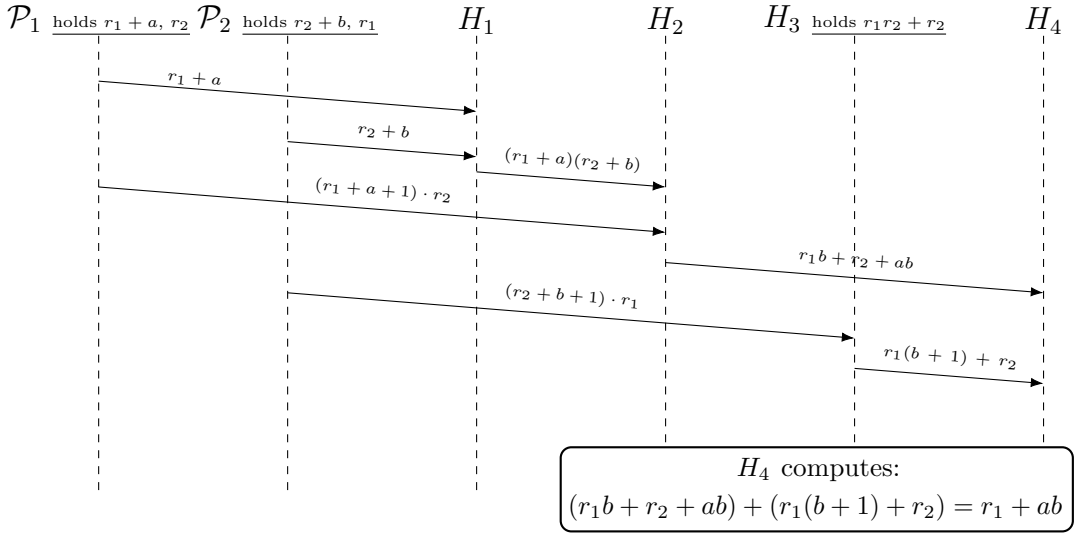
The helper roles have $(r_1, r_1 r_2 + r_3)$, it receives $b + r_2$, and sends $r_1(b + r_2) + (r_1 r_2 + r_3)$. For each fixed b, observe that the map

$$(r_1, r_2, r_3) \longmapsto (r_1, r_1 r_2 + r_3, b + r_2)$$

is a bijection on $\mathbb{F}_2^3$. The outgoing message is deterministic and independent of (a, b). □

2.3 Masked Multiplication (four helpers, two random bits)

Protocol Interaction Diagram: Masked Multiplication with four Helpers Π_4



This gadget computes ab while using only the two random bits r_1 and r_2. Its output is again masked by r_1, so it can be used as the input to a later gate.

The first helper party H_1 computes $(r_1 + a)(r_2 + b)$. Party $\mathcal{P}_1$ computes $(r_1 + a + 1)r_2$. Their sum is $r_1 b + r_2 + ab$. This follows from $(r_1 + a)(r_2 + b) + (r_1 + a + 1)r_2 = r_1 b + r_2 + ab$. Separately, $\mathcal{P}_2$ and H_3 compute $r_1(b + 1) + r_2$. When H_4 adds these two values, the r_2 terms cancel, and the final result is $r_1 + ab$.

Masked Multiplication with four Helpers $\Pi_4(\mathcal{P}_1, \mathcal{P}_2, (H_i)_{i\in\{1,2,3,4\}}, a, b, r_1, r_2)$

Input. $\mathcal{P}_1$ holds $r_1 + a$ and r_2 and $\mathcal{P}_2$ holds $r_2 + b$ and r_1. Helper H_3 holds $r_1 r_2 + r_2$. The other helpers receive only the messages specified below.

Output. H_4 holds $r_1 + ab$.

Protocol.

1. $\mathcal{P}_1 \to H_1$: send $r_1 + a$.
2. $\mathcal{P}_2 \to H_1$: send $r_2 + b$.
3. $H_1 \to H_2$: send $(r_1 + a)(r_2 + b)$.
4. $\mathcal{P}_1 \to H_2$: send $(r_1 + a + 1)\, r_2$.
5. $H_2 \to H_4$: send $(r_1 + a)(r_2 + b) + (r_1 + a + 1)r_2 = r_1 b + r_2 + ab$.
6. $\mathcal{P}_2 \to H_3$: send $(r_2 + b + 1)\, r_1$.
7. $H_3 \to H_4$: send $(r_2 + b + 1)r_1 + (r_1 r_2 + r_2) = r_1(b + 1) + r_2$.
8. H_4 outputs $(r_1 b + r_2 + ab) + (r_1(b + 1) + r_2) = r_1 + ab$.

Figure 3: Π_4: Secure delegated multiplication with two random bits (with 4 helpers)

Lemma 2.3. *For every player role in Figure 3, its view is distributed independently of* (a, b).

Proof. Let (a, b) be two inputs to the multiplication protocol and r_1, r_2 to be the two uniform random bits. We show that for every player role, the messages that it sends and receives are independent of (a, b). Specifically, P_1 view is $(r_2, U, (U+1)r_2)$ where $U = r_1 + a$, which is independent of (a, b). As in the previous proof, the pair (r_2, U) is uniform, and the last coordinate is a deterministic function of the first two arguments. The argument for P_2 is the same. If $V = r_2 + b$, then its view $(r_1, V, (V+1)r_1)$ is uniform over its first two coordinates, with the third determined by the first two. For H_1, the received values are $r_1 + a$ and $r_2 + b$. This is just an affine translation of random bits, so it is uniform. For H_2 let $U = (r_1 + a)(r_2 + b)$, and $V = (r_1 + a + 1)r_2$ H_2 view is $(U, V, U + V)$. Enumerating over all four choices for the two random bits gives

$$\Pr[(U, V, U + V) = (0, 0, 0)] = \frac{1}{2},$$

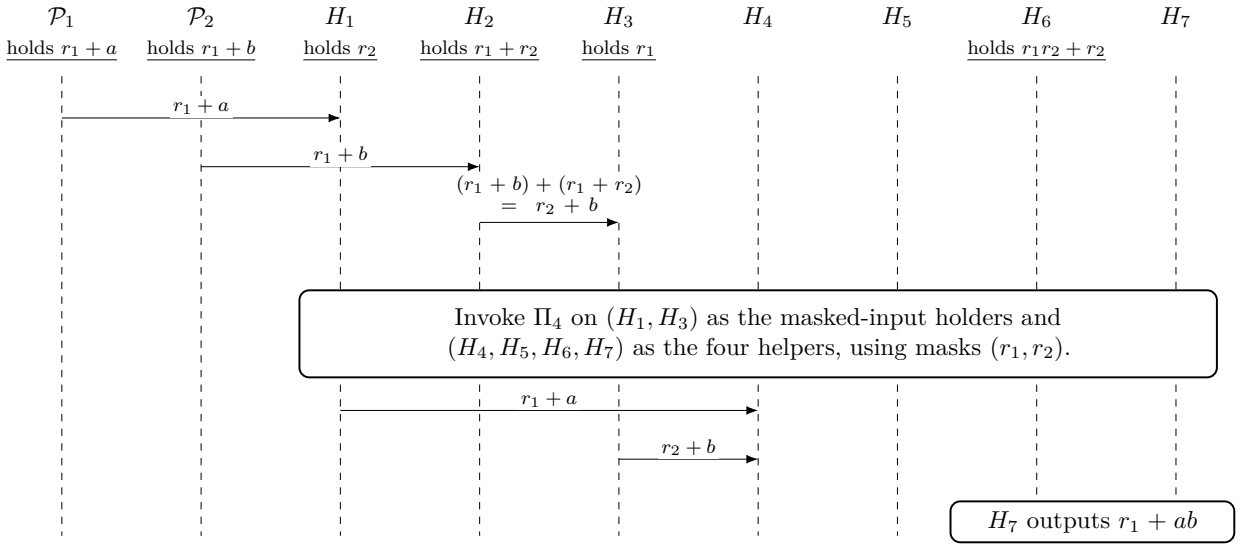
and probability $\frac{1}{4}$ for $(0, 1, 1)$ and $(1, 0, 1)$ and no other possible triples with positive probability. Observe that this is again independent of (a, b).

For H_3, set $L = r_1 r_2 + r_2$, $I = (r_2 + b + 1)r_1$, and $O = L + I$. With this notation, the view of H_3 is (L, I, O). It contains no occurrence of a, and for both choices of b has exactly the same distribution as for H_2.

Finally, H_4 receives $U = r_1 b + r_2 + ab$, and $V = r_1(b + 1) + r_2$. When $b = 0$, this becomes $(U, V) = (r_2, r_1 + r_2)$, and when $b = 1$, it becomes $(U, V) = (r_1 + r_2 + a, r_2)$. In both cases, the pair is uniform. Thus, every single role has a view whose distribution is independent of the inputs (a, b) □

2.4 Masked Multiplication with matching masks (seven helpers)

Protocol Interaction Diagram: Masked Multiplication with seven Helpers Π_7



Notice that we start with both inputs being masked by the same r_1. In the overall circuit protocol, this wrapper is invoked only in the distinct helper mode, separate from the above preprocessing. The Π_4 protocol needs the second input to be masked by r_2. Therefore, we first convert $r_1 + b$ into $r_2 + b$. Helper H_2 holds $r_1 + r_2$ and can compute $(r_1 + b) + (r_1 + r_2) = r_2 + b$. The parties then invoke Π_4 on the pair $r_1 + a$ and $r_2 + b$. We stress that Π_7 does not generate any new randomness.

Masked Multiplication with seven Helpers $\Pi_7(\mathcal{P}_1, \mathcal{P}_2, H^1, \ldots, H^7, a, b, r_1, r_2)$

Input. $\mathcal{P}_1$ holds $r_1 + a$ and $\mathcal{P}_2$ holds $r_1 + b$. Helpers: H_1 has r_2, H_2 has $r_1 + r_2$, H_3 has r_1, and H_6 has $r_1 r_2 + r_2$.

Output. H_7 gets $r_1 + ab$.

Protocol.

1. $\mathcal{P}_1 \to H_1$: send $r_1 + a$.
2. $\mathcal{P}_2 \to H_2$: send $r_1 + b$.
3. $H_2 \to H_3$: send $(r_1 + b) + (r_1 + r_2) = r_2 + b$.
4. Call
$$\Pi_4\big(H_1, H_3, H_4, H_5, H_6, H_7, a, b, r_1, r_2\big)$$
using masks r_1, r_2 with (blinded) inputs a and b. In Π_4, H_1 plays the role of $\mathcal{P}_1$ with $(r_1 + a, r_2)$ and H_3 plays the role of $\mathcal{P}_2$ with inputs $(r_2 + b, r_1)$. H_6 serves as provider of $r_1 r_2 + r_2$. Finally, H_7 receives the output.

Figure 4: Π_7: Multiplication with two random bits

Lemma 2.4. *For every player role in Figure 4, the party's view is independent of (a, b).*

Proof. $\mathcal{P}_1$'s local view consists only of the sent masked value $r_1 + a$. Similarly, $\mathcal{P}_2$'s local view consists of the masked value $r_1 + b$. Helper H_2 holds $r_1 + r_2$, receives $r_1 + b$, and sends $r_2 + b$ to H_3. Its view is $(r_1 + b, r_1 + r_2, r_2 + b)$, where the third coordinate is a deterministic function of the first two coordinates. The first two coordinates are uniform. Thus, every prefix view is independent of (a, b).

For parties that appear only in Π_4 call, Lemma 2.3 applies. The only roles that appear both before and during the internal call of Π_4 are H_1 and H_3. For H_1, its view is the first masked-input role of Π_4 with local data $(r_1 + a, r_2)$. H_1 also gets $r_1 + a$ from $\mathcal{P}_1$ before the internal call. H_3, view is the second masked-input role of Π_4 with local data $(r_2 + b, r_1)$. It also sees $r_2 + b$ received from H_2 before the internal call. Appending deterministic copies cannot change the distribution. Therefore, every single-party view in Π_7 is independent of (a, b). □

3 1-Private computation with helper parties

The construction below (with 1-privacy) collapses the randomness of arbitrary Boolean circuits to the optimal value of two bits, at the cost of $O(|C|)$ one-time helpers that do not

have any inputs themselves. This is orthogonal to the higher-threshold $t > 1$ collusion studied by Goyal, Ishai and Song [GIS22].

We evaluate the circuit gate by gate, where for every original wire value w, we will have a helper party holding it as a masked value $r_1 + x_w$, with the same masked bit r_1. This immediately creates a problem for the XOR gate, as direct addition would remove this mask. Instead, our protocol uses Π_3 to "remask" the sum. For the AND gate, our protocol uses Π_7. We note that all helper randomness used by these protocols is a deterministic function of the same two random bits (r_1, r_2) produced by the dealer. Specifically, the dealer (without any inputs of his own) picks two random bits and distributes the protocol-specific functions of these two bits to other players. Each gate of the original circuit is assigned a constant number of fresh helper parties to evaluate each gate. That is why local independence, while recycling the same randomness, is composable.

3.1 Secure Protocol with Helper Parties

We convert circuits to $\{\oplus, \wedge\}$ basis with public constants. Before the protocol starts, every wire value is represented as (w, τ) where τ is a public "switch" command, so that the actual wire value is $w + \tau$. In other words, when $\tau = 0$, it is the original value w of the wire. The negation gate is implicitly represented as $\tau = 1$, so that when the wire value is computed, it is publicly negated. Every remaining binary gate has two distinct input wires and two distinct current holders for these input wires.

Only the dealer tosses two random bits and sends the resulting combination to each party, as prescribed by the protocol. The remaining parties are deterministic. The protocol evaluates the circuit in topological order, using Π_3 for XOR gates and Π_7 for AND gates. At the end of the protocol, an input-party holder unmasks directly. If the final holder is a helper, it sends the public toggle of the masked output to $\mathcal{P}_1$, who unmasks and broadcasts.

Each input party $\mathcal{P}_i$ has input a_i receives a mask r_1 and on an input wire w it sends on the wire value $M + \tau_w = r_1 + x_w + \tau_w$ where τ_w is public and indicates whether this particular wire bit should be toggled or not.

Lemma 3.1 (Masked-wire fanout)**.** *For some wire w_j, assume a party contains a masked value $M_j = r_1 + x_{w_j}$. This party will only send deterministic functions of M_j. Therefore, the simulator can simulate these functions perfectly.*

Proof. Our simulator already samples the party's local view, including M_j. Later messages are deterministic functions of these values and the circuit description. Given the output, the final broadcast message is also fixed. Since adding deterministic values does not change the distribution, the simulated and real views are identical. □

3.2 One random bit is not sufficient — even with Helpers

We consider helper parties without any input as additional communication parties. We use the one-random-bit characterization and normal-form lower bound of Kushilevitz, Ostrovsky,

Secure helper-assisted computation with two random bits

Parties/Input. Input parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ hold private inputs $a_1, \ldots, a_n$. All operations are in $\mathbb{F}_2$ using only XOR and AND gates For circuit C let $k_\oplus$ be the number of XOR gates, and $k_\wedge$ be the total number AND gates. Before the protocol begins, we (syntactically) simplify C as follows: we remove gates where one of the input wires is a fixed constant (using our τ notation as needed). We also eliminate any gate where the two inputs come from a single input wire. As a result, every invoked gate has two separate input wires $(u, \tau_u), (v, \tau_v)$ with $u \neq v$. We allocate three helper parties H^1, H^2, H^3 for each XOR gate, and seven helper parties $H^1, \ldots, H^7$ for each AND gate. We stress that each gate is allocated entirely new helper parties that are exclusive to that gate. We allocate a separate dealer party that tosses two random bits and distributes combinations of these two bits to all other parties as needed. Therefore, the total number of helper parties is $1 + 3k_\oplus + 7k_\wedge$.
Output. All parties receive circuit output $C(a_1, \ldots, a_n)$.
Protocol.

1. **Dealer.** The dealer is the only party that tosses coins in our protocol. Specifically, it uniformly samples (r_1, r_2) and sends r_1 to all input parties. For XOR gates, recall that there are three helper parties, and the dealer sends $(r_2, r_1 + r_2)$ to (H^1, H^3). Recall that there are seven helper parties for each AND gate, and the dealer sends $(r_2, r_1 + r_2, r_1, r_1 r_2 + r_2)$ to (H^1, H^2, H^3, H^6).
2. **Masked wires.** Recall that we denote by x_w the value on wire w. Let $M_w = r_1 + x_w$. By $\mathrm{hold}(w)$, we denote the party that holds M_w. (Observe that for input wires w_i, $\mathrm{hold}(w_i) = \mathcal{P}_i$ and $\mathcal{P}_i$ can compute $M_{w_i} = r_1 + a_i$.)
3. **Gates.** We will evaluate the gates in topological order, from inputs to outputs. For gate o, let $o = ((u, \tau_u), (v, \tau_v), w, \oplus)$ or $o = ((u, \tau_u), (v, \tau_v), w, \wedge)$ and let $u \neq v$ be the corresponding input wires for o. The two operand values for this gate are therefore $x_u + \tau_u$ and $x_v + \tau_v$.
In case of $\oplus$-gate, call $\Pi_3(\mathrm{hold}(u), \mathrm{hold}(v), \mathbf{H}_o, x_u + \tau_u, x_v + \tau_v, r_1, r_2)$ and set $\mathrm{hold}(w) = H^3_o$.
In case of $\wedge$-gate invoke $\Pi_7(\mathrm{hold}(u), \mathrm{hold}(v), \mathbf{H}_o, x_u + \tau_u, x_v + \tau_v, r_1, r_2)$ and set $\mathrm{hold}(w) = H^7_o$.
Observe that in all cases $\mathrm{hold}(w)$ is the value $M_w = r_1 + x_w$.
4. **Output.** Let the final operand of the function evaluation be (w_f, τ_f). If $\mathrm{hold}(w_f)$ is held by some input party, it sends $(M_{w_f} + \tau_f) + r_1$ to all parties. Otherwise, $\mathrm{hold}(w_f)$ sends $M_{w_f} + \tau_f$ to $\mathcal{P}_1$, who sends $(M_{w_f} + \tau_f) + r_1 = C(a_1, \ldots, a_n)$ to all other parties.

Figure 5: Gate by Gate computation with Helpers using two random bits

Prouff, Rosén, Thillard, and Vergnaud [KOP+21, Theorem 4.7]. The exact consequence needed here is the following:

Lemma 3.2 (KOPRTV one-bit obstruction)**.** *For Boolean input parties, the one-bit normal form of [KOP+21, Theorem 4.7] cannot compute* AND_n *when* $n \geq 3$.

Proof. This is the public-output Boolean-domain impossibility from [KOP+21, Theorem 4.7]. □

Helper can be modeled as an ordinary party whose input is fixed to $\{\star\}$. Such a party remains in the transcript. It does not contribute any input to the normal form.

Lemma 3.3 (1-bit impossibility via the KOPRTV normal form)**.** *Set* $n \geq 3$, $m \geq 0$. *We define function* $F_{n,m}$ *as follows:* $F_{n,m} : \{0,1\}^n \times \{\star\}^m \to \{0,1\}$ *ignores the last* m *inputs and computes* $AND(a_1, \ldots, a_n)$. *Then, it is impossible to compute* $F_{n,m}$ *with perfect* 1*-privacy using only one random bit.*

Proof. Assume, for contradiction, that such a protocol exists. If it uses no random bit, add one unused uniform random bit. Thus, we may assume that it uses exactly one random bit.

Replace each singleton party by a Boolean dummy party with input $z_\ell \in \{0,1\}$. The dummy party runs the same algorithm as before and ignores z_ℓ. The functionality also ignores z_ℓ. Hence we obtain a one-random-bit protocol for the Boolean function $g(a_1,\ldots,a_n,z_1,\ldots,z_m) = \bigwedge_{i=1}^{n} a_i$.

Correctness is unchanged. Perfect privacy is unchanged as well: the original simulators handle the parties holding a_i, while the simulator for a dummy party ignores z_ℓ and uses the old singleton-party simulator.

By [KOP+21, Theorem 4.7], this Boolean-domain protocol has an equivalent one-bit normal-form representation for g. Now fix $z_1 = \cdots = z_m = 0$. After substituting these fixed values, we obtain a one-bit normal form for $g(a_1,\ldots,a_n,0,\ldots,0) = \mathrm{AND}_n(a_1,\ldots,a_n)$.

However, Lemma 3.2 says that this is impossible, a contradiction. □

Corollary 3.4 (Two random bits are necessary for helper-party AND)**.** *For every $n \geq 3$, any perfectly 1-private AND function, even with arbitrarily many helper parties without any inputs, must use at least two uniform random bits.*

Proof. Let a protocol use m helpers. This m is finite in any finite protocol. Lemma **??** rules out protocols with at most one uniform random bit for this fixed m. Since m was arbitrary, the claim follows. □

Remark 3.1. The lower bound is used only to rule out one random bit in the worst case, and the case $n = 3$ already suffices.

3.3 Sufficiency of Two Random Bits with Helpers

We now prove our Theorem 1.1.

Proof. For the upper bound, order circuit gates in a fixed topological order, and denote, for circuit C, the total number of gates as k. Each input is held as $r_1 + x_w$. We prove the correctness by induction on this fixed gate order. Consider a gate with inputs (u,τ_u) and (v,τ_v). The parties with the input to that gate send $r_1 + x_u + \tau_u$ and $r_1 + x_v + \tau_v$. For the XOR gate, we execute Π_3, which computes $r_1 + (x_u + \tau_u + x_v + \tau_v)$. For the AND gate, we execute Π_7 which computes $r_1 + ((x_u + \tau_u)(x_v + \tau_v))$. Hence, the invariant holds for every circuit wire. We unmask the final result, which reveals $C(a_1,\ldots,a_n)$.

To show privacy for any party X (without collusion) we show a perfect simulator for the view of X.

If X is a dealer party, its view consists of (r_1, r_2) and deterministic outgoing messages to other parties. Hence, the simulator samples (r_1, r_2) and computes the same messages.

If $X = \mathcal{P}_i$, the simulator is given $(a_i, C(\mathbf{a}))$. It samples $r_1 \leftarrow \mathbb{F}_2$ and outputs the dealer message r_1. It outputs deterministic messages prescribed by the protocol. These messages are determined by a_i, r_1, and the public toggle τ for each outgoing message.

When $i = 1$ and $\mathcal{P}_1$ receives the final masked output, the simulator outputs $r_1 + C(\mathbf{a})$. For parties other then $\mathcal{P}_1$, where $\text{hold}(w_f) = \mathcal{P}_i$, the simulator appends the public broadcast $C(\mathbf{a})$.

Now suppose X is a helper party that is not the output holder of its gate. This helper party participates in only a single gate computation. Its randomness is derived from a message from the Dealer. If the gate is an XOR gate, Lemma 2.1 simulates its gadget transcript. If the gate is an AND gate, Lemma 2.4 does the simulation.

Now, consider the helper party for gate o that holds the (masked) output of that gate. This helper is H^3 for an XOR gate and H^7 for an AND gate. For the computation of messages within each gate, we use simulations from Lemma 2.1 for XOR AND Lemma 2.4 for AND gates. This view includes the masked output $M = r_1 + x_w$. If this gate feeds into multiple other gates, consider fan out wires one by one. Sometimes these fan-out wires have to be toggled – but this is public information from the circuit description. Therefore, all fan-out messages for a gate are deterministic values of the output of that gate.

Every party also receives the final output of the computation. All of these messages can be simulated. Thus, every helper's view is simulatable.

Furthermore, observe that our construction uses exactly $1 + 3k_{\oplus} + 7k_{\wedge}$ helper parties for any circuit computation together with exactly two-bits of seed generated by the dealer party.

For the lower bound, use Corollary 3.4. It shows that an n-party AND with $n \geq 3$ cannot be computed with at most one uniform random bit, even with helpers. Since we have shown a protocol to compute any function 1-privately with exactly two random bits, the helper-party randomness complexity is exactly two bits (for every input length $n \geq 3$). □

3.4 KOR equivalence even with helpers

Kushilevitz, Ostrovsky, and Rosén [KOR96, KOR99] showed that any 1 private protocol for a function f that uses only a constant number of random bits implies that f has a linear-size circuit. In this section, we extend this result to 1-private protocols even in the presence of helper parties.

Lemma 3.5 (Kushilevitz–Ostrovsky–Rosén protocol-to-circuit reduction)**.** *If there exists a perfectly* 1*-private semi-honest protocol over private point-to-point channels for a Boolean function on N bits, where each protocol participant receives as his private input exactly one input bit, and the protocol uses d random bits, then this function has a Boolean circuit of size $2^{O(d)}N$.*

Proof. In [KOR96] (also, see the journal version, [KOR99, Lemma 3]), consider a 1-private protocol, where they encode messages as bits, enumerate over all possible random bit strings, and construct a circuit from the protocol. The stated size bound is $2^{O(d)}N$. Our protocol fits their format exactly. Specifically, dealer messages to parties is communication, not additional randomness, and $d \leq 2$. □

Proof of Corollary 1.2. Let's consider any Boolean function on n input bits. If there exists a circuit (over XOR and AND gates) with $O(n)$ gates, then Theorem 1.1 evaluates these circuits with at most two random bits and $O(n)$ helper parties.

Conversely, from the upper bound of our theorem 1.1, we already have a perfectly 1-private helper-party protocol using at most two random bits and $m(n) = O(n)$ helper parties without input. Define $g_n(x_1, \ldots, x_n, z_1, \ldots, z_{m(n)}) = f_n(x_1, \ldots, x_n)$. Thus, function g_n ignores the dummy inputs z. We now turn each helper party (without input) into an ordinary input party with a Boolean dummy input z_j. This input is ignored by the functionality, by the real protocol, and by the simulator. Perfect privacy is therefore preserved. Hence, g_n has a no-helper perfectly 1-private protocol using at most two random bits. Lemma 3.5 gives circuits for g_n of size $2^{O(1)}(n + m(n)) = O(n)$. Hardwiring the dummy inputs gives circuits for f_n of size $O(n)$. □

4 Using a 5-SWIG without helpers

We now prove Theorem 1.3 from $(5, n-1)$-SWIG. We use the one-helper protocol Π_1, shown in Figure 2, as a building block. Observe that in the no-helper setting, every helper role must be played by an input party. The difficulty is that a real party must participate in multiple roles in this protocol. Nevertheless, every party view must still look as if all masks in all roles of this party use fresh random bits.

Let $I_n = \{2, 3, \ldots, n\}$. We use indices in I_n, where $i + d$ is interpreted with wraparound, so that $i + d$ is treated as $2 + ((i - 2 + d) \bmod (n - 1))$. We fix players in an arbitrary order and then compute prefixes of multiplication. That is, multiplication of step i assumes that we have already (privately) multiplied numbers up to $i - 1$, and party $\mathcal{P}_{i-1}$ has that previous blinded multiplication result. On the other hand, $\mathcal{P}_i$ initially acts as a second party with its own private input a_i, and the role of the helper party will be played by $\mathcal{P}_{i-2}$. That is, for a fixed $\mathcal{P}_i$ with $i \in I_n$, he plays the following three roles: he plays the second input role at step i (with input a_i), the first input role at step $i + 1$ if $i < n$ (for $i = n$ transmitting the final result back to $\mathcal{P}_1$), and in a helper role at step $i + 2$. We stress that these roles are distinct for $|I_n| \geq 5$.

This warm-up is only a guide to the mask schedule and is not used as the actual proof. Its correctness follows from the same induction as the full protocol with independent masks.

The warm-up uses an independent mask for each position in the multiplication chain. The SWIG version keeps the same schedule, but it replaces the independent masks by sliding windows coming from the SWIG generator. Our SWIG schedule replaces the triple (r_i, r_{i-1}, r_{i+1}) at step i with (z_i, z_{i+2}, z_{i+1}).

4.1 The SWIG schedule

Recall that $I_n = \{2, 3, \ldots, n\}$ which we treat cyclically. We will use a $(5, n-1)$-SWIG, which provides a generator $G_n : \mathbb{F}_2^5 \to \mathbb{F}_2^{I_n}$ where we set a seed S to be five random bits and denote

$$G_n(S) = (z_i)_{i \in I_n}$$

Recall that by the 5-SWIG property, every cyclic block of length five $(z_i, \ldots, z_{i+4})$ is uniform, with wrap-around. Now, consider step i. This step uses z_i to mask the AND-prefix of up

Warm-Up: Sequential Composition (not used in final construction)

Input. n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ have private inputs $a_1, \ldots, a_n$.

Output. All parties get $\prod_{j=1}^{n} a_j$.

Protocol.

1. $\mathcal{P}_1$ in addition to his usual role, also acts as a dealer, picking uniform random bits $(r_1, \ldots, r_{n+1})$.
2. $\mathcal{P}_1$ distributes r_{t-1}, r_{t+2}, and $r_{t+2}r_{t+1} + r_{t+3}$ to $\mathcal{P}_t$ for $2 \le t \le n-2$.
3. Finally, $\mathcal{P}_1$ also sends r_{n-2}, r_2, and $r_2 r_1 + r_3$ to $\mathcal{P}_{n-1}$. It sends r_{n-1}, r_3, and $r_3 r_2 + r_4$ to $\mathcal{P}_n$.
4. For $i = 2, 3, \ldots, n$, in increasing order, parties execute:

$$\Pi_1\left(\mathcal{P}_{i-1}, \mathcal{P}_i, \mathcal{P}_{i-2}, \prod_{j=1}^{i-1} a_j, a_i, r_i, r_{i-1}, r_{i+1}\right)$$

   By roles, we mean the order of arguments in the Π_1 protocol invocation. That is, when a player is called as the first argument to Π_1, we say that he plays the "first" role, and so on. In our calls to Π_1, the first role player holds a smaller prefix result $(\prod_{j=1}^{i-1} a_j) + r_i$ of the multiplication. For the first invocation, when $i = 2$, the first role is just player $\mathcal{P}_1$ holding $a_1 + r_1$. The second role uses his own private input a_i and mask r_{i-1}. The third, helper role uses setup bits r_i and $r_i r_{i-1} + r_{i+1}$. The very first call to $\Pi + 1$ is an input-holder variant of Π_1. The rest of the calls to Π_1 are masked-holder variants of Π_1 for all $i > 2$.
5. Finally, $\mathcal{P}_n$ sends $(\prod_{j=1}^{n} a_j) + r_{n+1}$ to $\mathcal{P}_1$.
6. $\mathcal{P}_1$ unmasks the message from $\mathcal{P}_n$ using r_{n+1} and sends $\prod_{j=1}^{n} a_j$ to all parties.

Figure 6: n-party AND with $n+1$ bits of randomness (baseline)

to $\mathcal{P}_{i-1}$. Player $\mathcal{P}_i$ uses z_{i+2} as the mask for his own input a_i, and uses z_{i+1} as the output mask. Observe that the output mask at step i is exactly the incoming mask at step $i+1$. The dealer sends the following pair to player $\mathcal{P}_i$ of values:

$$(z_{i+2}, \quad z_{i+2}z_{i+4} + z_{i+3})$$

This pair of random bits has two roles. $\mathcal{P}_i$ plays the role of second-party at step i and uses the first bit as his mask. The same pair of bits is also used by $\mathcal{P}_i$ as a helper party during step $i+2$, because that later step needs helper-role values z_{i+2} and $z_{i+2}z_{i+4} + z_{i+3}$.

Definition 4.1 (Randomness complexity for AND without helpers)**.** *For each $n \ge 2$, let β_n denote the minimum number of uniform random bits required by any perfectly 1-private semi-honest no-helper protocol for n-party AND, allowing arbitrary communication patterns and arbitrary numbers of rounds.*

Lemma 4.2. *Let $n \ge 6$. If there is a $(5, n-1)$-SWIG over $\mathbb{F}_2$, then the protocol in Figure 7 is correct and perfectly 1-private. Thus, $\beta_n \le 5$.*

4.2 The 5-bit protocol

The protocol follows the warm-up multiplication chain. All masks are now coordinates of the SWIG output. The proof uses only the exact local uniformity of length-five cyclic windows.

Full n-party AND Protocol

Input. n parties $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ hold bits $a_1, \ldots, a_n$.

Output. $\prod_1^n a_i$.

Protocol.

1. $\mathcal{P}_1$ samples $S \leftarrow \mathbb{F}_2^5$ uniformly and computes the $(5, n-1)$-SWIG: $G_n(S) = (z_i)_{i \in I_n}$.
2. $\mathcal{P}_1$ sends to $\mathcal{P}_{2 \le i \le n}$ the following SWIG bits: $(z_{i+2},\ z_{i+2}z_{i+4} + z_{i+3})$.
3. For $i = 2, 3, \ldots, n$ in increasing order, parties $(\mathcal{P}_{i-1}, \mathcal{P}_i, \mathcal{P}_{\sigma(i)})$ execute Π_1:

$$\Pi_1\left(\mathcal{P}_{i-1}, \mathcal{P}_i, \mathcal{P}_{\sigma(i)}, \prod_{j=1}^{i-1} a_j, a_i, z_i, z_{i+2}, z_{i+1}\right)$$

   For each Π_1 call, $\mathcal{P}_{i-1}$ uses as his input the running product $(\prod_{j=1}^{i-1} a_j) + z_i$. For the very first call to Π_1 with $i = 2$, since $\mathcal{P}_{i-1} = \mathcal{P}_1$, $\mathcal{P}_1$ uses his input-holder variant of Π_1. In the second role of Π_1 invocation, party $\mathcal{P}_i$ uses his input a_i and mask z_{i+2}. The helper role $\mathcal{P}_{i-2}$ uses SWIG bits z_i and $z_i z_{i+2} + z_{i+1}$.
4. $\mathcal{P}_n$ sends $(\prod_{j=1}^{n} a_j) + z_{n+1}$ to $\mathcal{P}_1$.
5. Finally, $\mathcal{P}_1$ unmasks $\mathcal{P}_n$'s with z_{n+1} and then sends the ouput $\prod_{j=1}^{n} a_j$ to all other parties.

Figure 7: Full n-party AND protocol with 5-bit seed and without helpers.

4.3 Proof of Lemma 4.2

We will describe a party's simulated view by grouping its messages according to the roles it plays. This is only a notational convenience to make our simulation argument easier to present. In the real protocol, messages are delivered in public chronological order: the setup messages come first, and then the calls to Π_1 are executed in increasing order. The time when the sender sends a particular message to the receiver is public and fixed for each player. Therefore, a fixed role-indexed transcript uniquely defines a chronological transcript as well.

The simulator may first sample the messages in the role order that is most convenient for the privacy argument, and this is what we do below. Before outputting such a view, the simulator reorders those messages according to chronological order. This reordering operates on a transcript and therefore does not introduce new randomness or change the distribution. Therefore, the role-based transcript and chronological transcript have the same distribution after re-ordering.

Proof of Lemma 4.2. For step i, let $h = \sigma(i) = i - 2$. $\mathcal{P}_1$, in his role as a dealer, sends to $\mathcal{P}_h$

SWIG-derived bits:

$$(z_{h+2}, \quad z_{h+2}z_{h+4} + z_{h+3}) = (z_i, \quad z_i z_{i+2} + z_{i+1}).$$

that $\mathcal{P}_h$ in the role of helper at step i requires.

We now prove correctness by induction on prefixes $i \leq n$ of the multiplication chain. For $i = 2$, $\mathcal{P}_1$ transmits $a_1 + z_2$ to $\mathcal{P}_2$. When we execute protocol Π_1, it outputs $a_1 a_2 + z_3$. Assume that the protocol is correct by induction up to $i < n$ The output of Π_1 after step i is held by $\mathcal{P}_i$, and it holds

$$\left(\prod_{j \leq i} a_j\right) + z_{i+1}.$$

This is the correct first-role input for step $i + 1$. Next application of Π_1 gives

$$\left(\prod_{j \leq i+1} a_j\right) + z_{i+2}.$$

At the n'th step, party $\mathcal{P}_n$ sends back the final masked product to $\mathcal{P}_1$. Party $\mathcal{P}_1$ unmasks the message from $\mathcal{P}_n$ and sends it to everyone.

We now prove (perfect) privacy. Let $f(\mathbf{a}) = \prod_{j=1}^{n} a_j$. We start by constructing a (perfect) simulator for $\mathcal{P}_1$. On input $(a_1, f(\mathbf{a}))$, the simulator samples a seed S consisting of five uniform random bits for 5-SWIG, and computes the $(5, n-1)$-SWIG output $G_n(S) = (z_i)_{i \in I_n}$. It also outputs a_1, the SWIG seed S, and all dealer messages to other players (derived from SWIG output). It also outputs $i = 2$ message $a_1 + z_2$ to $\mathcal{P}_2$. The simulator also outputs the last incoming message from $\mathcal{P}_n$ (equal to $f(\mathbf{a}) + z_{n+1}$) and final transmission to all other players with the output $f(\mathbf{a})$ of the computation. This perfectly $\mathcal{P}_1$'s view.

Fix $i \in I_n$. Let

$$(s_0, s_1, s_2, s_3, s_4) = (z_i, z_{i+1}, z_{i+2}, z_{i+3}, z_{i+4})$$

This is a sliding window of length 5 and hence uniform, since it comes from the 5-SWIG. For $i < n$, party $\mathcal{P}_i$ has three roles: the second input party at step i, the first input party at step $i + 1$, and the helper at step $t = i + 2$ (recall that t treated cyclically in I_n). For $i = n$, the first role message at the next step is replaced by the final send from $\mathcal{P}_n$ to $\mathcal{P}_1$.

We already specified the simulator for $\mathcal{P}_1$. Now, we specify simulators for other players (with $\mathcal{P}_n$ treated separately). Let $p_{i-1} = \prod_{j<i} a_j$ and $q = s_2 s_4 + s_3$. Suppose first that $i < n$. The simulator samples random bits s_2, q, M, H, X independently and uniformly. It records the setup pair (s_2, q), the incoming step-i message M, the second-role outgoing message $a_i + s_2$, the helper reply H, the next first-role message $M' = a_i M + H$ and the helper-role messages X and $Y = s_2 X + q$.

The last party, $i = n$, is treated separately because its next first-role message is replaced by the final send. In this case, the simulator samples bits s_1, s_2, q, M, X independently and uniformly. It records the setup pair (s_2, q), the incoming step-n message M, and the outgoing second-role message $a_n + s_2$. It then sets $M' = f(\mathbf{a}) + s_1$. The value of H is forced by the relation $H = a_n M + M'$. Equivalently, if $a_n = 0$ it sets $H = M'$, while if $a_n = 1$ it

sets $H = M + M'$. It records H, the final send M', and the helper-role messages X and $Y = s_2X + q$.

In all cases, the simulator also appends the function output $f(\mathbf{a})$ message from $\mathcal{P}_1$.

Since we run the simulation in the role-based view, we can also compare it to the real execution in the role-based view. Now, Fix partial product p_{i-1} and input a_i. In the real protocol, the two visible messages come from the two mask bits (s_0, s_1) as:

$$(M, H) = (p_{i-1} + s_0, \quad a_i s_0 + s_1)$$

Notice that this is just an affine change of variables. In other words, every choice of (s_0, s_1) gives exactly one pair (M, H), and every pair (M, H) comes from exactly one choice of (s_0, s_1). Since (s_0, s_1) is uniform, (M, H) is uniform as well.

Here, by the setup pair we mean the two setup values (s_2, q). The value s_2 is one of the setup bits, and q is the derived bit given by $q = s_2s_4 + s_3$. Now suppose these two values, s_2 and q, have already been fixed. Even then, s_4 has not been determined. For either possible value of s_4, there is exactly one choice of s_3 that makes the equation $q = s_2s_4 + s_3$ true. So fixing (s_2, q) does not make one value of s_4 more likely than the other. The bit s_4 is still uniform.

Because a_t is fixed, the value $X = a_t + s_4$ is also uniform. Adding the fixed bit a_t only possibly flips 0 and 1; it does not change the fact that the value is random. Once X is known, however, the helper's other message is no longer random. It is determined by the formula $Y = s_2X + q$. Thus the helper role has the same behavior in the simulation as it does in the real protocol: first X is uniformly distributed, and then Y is forced by the already fixed setup values.

Finally, the randomness used for (M, H) is independent of this remaining choice of s_4. So the real second-role and helper-role messages have exactly the same joint distribution as the simulator's messages. Hence the real and simulated role-indexed views are identically distributed. Reordering the same messages into chronological order does not change this conclusion, so the chronological views are identically distributed as well. □

5 Explicit Construction of k-SWIGs

5.1 Non-constructive proof of SWIG existence

For completeness, we first present a self-contained existence proof. This proof essentially derives the cyclic, fixed-basis specialization of the interpolation-between-bases theorem of Linial and Tarsi [LT89]. In rank-matrix language, the condition is closely related to the binary n-regular matrices of Etzion and Lempel [EL86].

Using the equivalence established in Lemma 1.8, we show that there exists a sequence of m vectors such that every sliding window of length k is linearly independent, treating indices mod m. First, note that if $e_1, \ldots, e_k$ form a basis for $\mathbb{F}_2^k$, then $e_1, \ldots, e_k$ is a (k, k)-SWIG. We show how to convert any (k, m)-SWIG, for $m \geq k$, into a $(k, m + 1)$-SWIG. This conversion relies on the following:

Lemma 5.1. *Let $v_1, v_2, \ldots v_{2n}$ be vectors in $V = \mathbb{F}_2^{n+1}$. For $0 \le i \le n$, let $S_i = \{v_{i+1}, v_{i+2}, \ldots v_{i+n}\}$, and for $0 \le i \le n-1$, let $T_i = \{v_{i+1}, v_{i+2}, \ldots v_{i+n+1}\}$. Suppose that for all i, the set T_i is linearly independent. Then there exists a vector $w \in V$ such that the sets $S_i \cup \{w\}$ are all linearly independent.*

Proof. Each S_i is a subset of a linearly independent set, and so is itself linearly independent; it consequently generates a hyperplane H_i. Let p_i be the unique nonzero element of V^* such that $H_i = \ker p_i$. Each $S_i \cup \{w\}$ is linearly independent iff $w \notin H_i$ iff $p_i(w) = 1$.

For $0 \le i \le n-1$, the set T_i is $\{v_{i+1}\} \cup S_{i+1}$. Since T_i is linearly independent, v_{i+1} is not generated by S_{i+1}. Rephrasing, $v_{i+1} \notin H_{i+1}$, so $p_{i+1}(v_{i+1}) = 1$, i.e. $p_j(v_j) = 1$ for $1 \le j \le n$.

Suppose that $c_0 p_0 + c_1 p_1 + \cdots + c_n p_n = 0$ is a nontrivial relation on the p_i. Let c_ℓ be the nonzero coefficient of maximal index; since all $p_i \neq 0$, necessarily $\ell > 0$. We have $c_0 p_0(v_\ell) + c_1 p_1(v_\ell) + \cdots + c_\ell p_\ell(v_\ell) = 0$. Now $p_\ell(v_\ell) = 1$, while when $i < \ell$, the vector v_ℓ is in S_i, so $p_i(v_\ell) = 0$. The relation therefore becomes $c_\ell \cdot 1 = 0$, which is absurd. The p_i are therefore linearly independent.

Since there are $n+1$ vectors p_i, they form a basis of V^*. Setting $\varphi(p_i) = 1$ for all i then defines an element $\varphi \in V^{**}$. The canonical isomorphism $V^{**} \cong V$ implies that there exists a unique $w \in V$ such that $\varphi(p) = p(w)$ for all $p \in V^*$. We then have $p_i(w) = 1$ for all i. □

To prove the induction step, let $v_1, \ldots, v_m$ be a (k, m)-SWIG. Apply the preceding lemma with $n = k-1$; if $m < 2k-2$, then set $v_{m+1}, \ldots, v_{2k-2} = v_1, \ldots, v_{2k-2-m}$. The sets T_i from the lemma are sliding windows of length k, and so are linearly independent, by the induction hypothesis. Therefore we can apply the lemma to obtain a vector $w \in V$ such that the sets $\{v_1, \ldots, v_{k-1}, w\}, \{v_2, \ldots, v_{k-1}, w, v_k\}, \ldots, \{w, v_k, \ldots, v_{2k-2}\}$ are linearly independent. Inserting w between v_{k-1} and v_k gives a $(k, m+1)$-SWIG.

We now prove Theorems 1.5 and 1.6. The construction repeats the standard basis and then attaches the Pascal suffix. We will use the following binomial-coefficient identities in our proof:

5.2 Binomial Coefficient Checks

Lemma 5.2 (Newton Determinants)**.** *For integer-valued $x, h \ge 0$ and $r \ge 1$:*

$$\det\left[\binom{x+\beta}{\alpha}\right] = 1$$

and

$$\det\left[\binom{h+\alpha+\beta}{\beta}\right] = 1$$

where α and β range over $0 \le \alpha, \beta < r$. Over integer matrix entries, both determinants are 1. Furthermore, even after reducing entries modulo two, they both remain 1.

Proof. Observe that the first determinant evaluates the Newton basis polynomials $\binom{X}{0}, \ldots, \binom{X}{r-1}$ at $x, x+1, \ldots, x+r-1$. The second determinant evaluates the polynomials $\binom{h+X}{0}, \ldots, \binom{h+X+r-1}{r-1}$ at $0, 1, \ldots, r-1$. In both cases, the leading coefficients cancel the consecutive-point Vandermonde products. Therefore, in both cases, the determinant is 1. □

Lemma 5.3 (Complement identity modulo two)**.** *For any non-negative integer m, set $N = 2^m$. Then, for all non-negative integers x, y below N,*

$$\binom{N-1-x}{y} \equiv \binom{x+y}{y} \pmod 2.$$

Proof. Since $N = 2^m$, we have $N - 1 = 2^m - 1$, whose m low-order bits are all 1. Thus, $N - 1 - x$ can be obtained by flipping m low-order bits of x. We start by considering the left side of the identity. By Lucas's theorem, $\binom{N-1-x}{y}$ is odd exactly when every 1-bit of y also appears as a 1-bit of $N - 1 - x$. Since the 1-bits of $N - 1 - x$ are exactly where x has 0-bits among the m low-order positions, this means that x and y have no common 1-bit in their binary representation. Consider the right-hand side. By the no-carry condition for binomial coefficients mod two, we know that $\binom{x+y}{y}$ is odd exactly when adding x and y in binary representation produces no carries. This condition holds exactly when x and y in binary representation do not have a common 1 in any position. Observe that both sides are odd under exactly the same condition — namely that the binary representations of x and y do not have common 1's in the same position. If this condition fails, both sides are even. Hence, the two binomial coefficients are congruent modulo two. □

5.3 Pascal suffix construction

To the best of our knowledge, our SWIG construction is the first explicit, closed-form construction for binary cyclic windows. The broader consecutive-basis framework goes back to [LT89, EL86].

Fix $k \geq 1$ and $m \geq k$. Write $m = qk + s$, where $q \geq 1$ and $0 \leq s < k$. Let $e_1, \ldots, e_k$ be the standard basis of $\mathbb{F}_2^k$. Let N be the least power of two with $N \geq k$, and set $a = N - k$. If $k \geq 2$, define Pascal suffix vectors by coordinates as follows. For $0 \leq t \leq k-2$ and $1 \leq c \leq k$,

$$(\mathbf{b}_t)_c = \binom{a+c-1}{t} \bmod 2$$

For $k = 1$, there is no suffix. Let the coefficient vectors $\mathbf{u}_0, \ldots, \mathbf{u}_{m-1}$ consist of q copies of $e_1, \ldots, e_k$, followed by the first s Pascal suffix vectors $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{s-1} \in \mathbb{F}_2^k$. The suffix is empty when $s = 0$. We are ready to define the SWIG:

$$G_{k,m}(S)_i = \langle \mathbf{u}_i, S \rangle$$

Theorem 5.4 (Construction and proof of k-SWIG Theorem 1.6)**.** *For every $k \geq 1$ and every $m \geq k$, $G_{k,m}$ is a (k, m)-SWIG.*

Proof. By Lemma 1.8, it is sufficient to show that every block of k consecutive coefficient vectors $\mathbf{u}_i, \mathbf{u}_{i+1}, \ldots, \mathbf{u}_{i+k-1 \pmod m}$ has rank k. If $k = 1$, this is immediate. We now assume $k \geq 2$.

Any block contained entirely in the repeated standard-basis part is a basis. It remains to consider boundary blocks, namely those spilling into or out of the Pascal suffix. Let r be the number of suffix rows in such a block. Then $1 \leq r \leq s$.

There are two cases. First, a block may enter the Pascal suffix from the repeated standard basis part. In this case, after deleting the standard-basis pivot rows, the remaining $r \times r$ matrix has entries, for $0 \leq \alpha, \beta < r$,

$$\left[\binom{a+\ell+\beta}{\alpha}\right]$$

where $0 \leq \ell \leq k - r$. Its determinant is 1 by Lemma 5.2.

Second, a block may start in the Pascal suffix vectors and wrap around to the standard basis. There is some $0 \leq j \leq s - r$, such that the reduced matrix is

$$M_{\alpha,\beta} = \binom{N - r + \beta}{j + \alpha}$$

for $0 \leq \alpha, \beta < r$. Reverse the columns, set $x = r - 1 - \beta$. The entries therefore become

$$\binom{N - 1 - x}{j + \alpha}$$

and since $j + \alpha \leq k - 2 < N$, Lemma 5.3 gives, modulo 2,

$$\binom{N - 1 - x}{j + \alpha} = \binom{x + j + \alpha}{x}.$$

The second equality of Lemma 5.2 now applies and shows that all such determinants are 1. We have therefore now considered all possible cyclic length-k blocks. □

5.4 SWIG construction for $k = 5$

Proof of Theorem 1.5. We spell out the $(5, m)$-SWIG of Theorem 5.4.

For any $m \geq 5$ we first determine q and s where $m = 5q + s$, with $q \geq 1$ and $0 \leq s < 5$. We let $e_1, \ldots, e_5$ be standard basis of $\mathbb{F}_2^5$. We pick N such that $5 \leq N$ and is a power of two. That give $N = 8$ and $a = N - 5 = 3$. Following Theorem 5.4 for $0 \leq t \leq 3$, define

$$\mathbf{b}_t = \left(\binom{a}{t}, \binom{a+1}{t}, \binom{a+2}{t}, \binom{a+3}{t}, \binom{a+4}{t}\right) \pmod 2$$

Since $a = 3$, the suffix vectors become:

$$\mathbf{b}_t = \left(\binom{3}{t}, \binom{4}{t}, \binom{5}{t}, \binom{6}{t}, \binom{7}{t}\right) \pmod 2.$$

Since t goes from 0 to 3, the suffix vectors mod 2 become:

$$\mathbf{b}_0 = (1,1,1,1,1)$$
$$\mathbf{b}_1 = (1,0,1,0,1)$$
$$\mathbf{b}_2 = (1,0,0,1,1)$$
$$\mathbf{b}_3 = (1,0,0,0,1)$$

We now form the coefficient sequence by starting with q copies of basis vectors

$$e_1, e_2, e_3, e_4, e_5,$$

and finishing with Pascal suffix vectors:

$$\mathbf{b}_0, \ldots, \mathbf{b}_{s-1}$$

If $s = 0$, we append nothing to the basis vectors. This gives coefficient vectors $\mathbf{u}_0, \ldots, \mathbf{u}_{m-1} \in \mathbb{F}_2^5$. The 5-SWIG takes 5 random bits as a seed S, and the i'th bit of $(5, m)$-SWIG is

$$G(S)_i = \langle \mathbf{u}_i, S \rangle$$

□

6 Additional results

6.1 A limitation of the five-bit cyclic framework for $n = 5$

The next obstruction concerns whether there is a transformation-based formulation beyond SWIG. It is not a limitation of the abstract SWIG definition itself. The SWIG protocol requires output length $m = n - 1 \geq 5$, so it does not apply to $n = 5$. One might still try to keep five seed bits and change the helper assignment. The next definition captures the same transformation-based one-helper-per-step framework with an arbitrary helper permutation.

This definition is included to connect the $n = 5$ obstruction back to the AND protocol. The SWIG schedule in Section 4 is a generator-level sufficient condition for the protocol in Figure 7. It is not the weakest way to express the linear recycling needed by that protocol. A weaker protocol-specific formulation only asks that, after the helper assignment is chosen, each party's five relevant mask coordinates are uniform and correctly matched to the calls to Π_1. The next definition formalizes this relaxation.

Let $I_n = \{2, 3, \ldots, n\}$, where indexing $i + 1$ in I_n is defined with wrap-around.

Definition 6.1 (The 5-agreeable conditions)**.** *Integer $n \geq 2$ is* 5-agreeable *if there exists a permutation $\sigma : I_n \to I_n$ and invertible linear transformations $T_i : \mathbb{F}_2^5 \to \mathbb{F}_2^5$ such that for every $i \in I_n$ the following four conditions simultaneously hold:*

$$\textbf{(C1)} \quad T_i^\top \mathbf{e}_3 = T_{i+1}^\top \mathbf{e}_1$$
$$\textbf{(C2)} \quad T_{\sigma(i)}^\top \mathbf{e}_2 = T_i^\top \mathbf{e}_1$$
$$\textbf{(C3)} \quad T_{\sigma(i)}^\top \mathbf{e}_4 = T_i^\top \mathbf{e}_2$$
$$\textbf{(C4)} \quad T_{\sigma(i)}^\top \mathbf{e}_5 = T_i^\top \mathbf{e}_3$$

The protocol interpretation is as follows. Suppose n is 5-agreeable and let $S \leftarrow \mathbb{F}_2^5$. A transformation-based version of the protocol in Figure 7 uses $\langle T_i^\top \mathbf{e}_1, S\rangle$ as the incoming product mask at step i, $\langle T_i^\top \mathbf{e}_2, S\rangle$ as the input mask for a_i, and $\langle T_i^\top \mathbf{e}_3, S\rangle$ as the output mask. It uses $\mathcal{P}_{\sigma(i)}$ as the helper at step i. Party $\mathcal{P}_h$ receives the setup pair

$$\Big(\langle T_h^\top \mathbf{e}_2, S\rangle,\ \langle T_h^\top \mathbf{e}_2, S\rangle\langle T_h^\top \mathbf{e}_4, S\rangle + \langle T_h^\top \mathbf{e}_5, S\rangle\Big)$$

Condition (C1) enforces consistency of masks required between consecutive steps of our Π_1 protocol. Specifically, the output mask at step i is the next partial product mask at step $i+1$. The conditions (C2)–(C4) allow proper handling of helper masks. These conditions ensure that the setup stored by $\mathcal{P}_{\sigma(i)}$ is precisely the helper masks needed for the Π_1 call at step i.

Consider party $\mathcal{P}_i$, and let $t = \sigma^{-1}(i)$ be the step in which $\mathcal{P}_i$ plays the helper role. Set

$$\begin{aligned} s_0 &= \langle T_i^\top \mathbf{e}_1, S\rangle \\ s_1 &= \langle T_i^\top \mathbf{e}_3, S\rangle \\ s_2 &= \langle T_i^\top \mathbf{e}_2, S\rangle \\ s_3 &= \langle T_i^\top \mathbf{e}_5, S\rangle \\ s_4 &= \langle T_i^\top \mathbf{e}_4, S\rangle \end{aligned}$$

Now, observe that since T_i is invertible, the tuple $(s_0, s_1, s_2, s_3, s_4)$ preserves full rank and is therefore uniform since T_i is invertible.

The entries s_0, s_1, s_2 are the incoming product mask, output mask, and input mask received by $\mathcal{P}_i$ at step i. Consider step t for the helper party. Then, conditions (C2)–(C4) identify s_2, s_4, s_3 as conditions on the previous AND-prefix mask, local input mask, and the output mask required by our Π_1 protocol. Thus, with these conditions, the local view of $\mathcal{P}_i$ has the same form as in Lemma 4.2. Therefore, the same simulator applies, and 5-agreeability is sufficient for the same one-helper-per-step AND construction.

The SWIG schedule used in Section 4 is a special case. If $z_i = \langle \mathbf{u}_i, S\rangle$ and $\sigma(i) = i-2$, define

$$\begin{aligned} T_i^\top \mathbf{e}_1 &= \mathbf{u}_i \\ T_i^\top \mathbf{e}_2 &= \mathbf{u}_{i+2} \\ T_i^\top \mathbf{e}_3 &= \mathbf{u}_{i+1} \\ T_i^\top \mathbf{e}_4 &= \mathbf{u}_{i+4} \\ T_i^\top \mathbf{e}_5 &= \mathbf{u}_{i+3} \end{aligned}$$

The SWIG condition gives invertibility of each T_i. The four identities above allow us to recycle bits of the protocol in Figure 7. Thus, our 5-agreeability definition relaxes and is more general than the SWIG conditions. 5-agreeability allows other helper permutations, and it does not require a single cyclic output sequence with all sliding windows to be uniform. Unfortunately, Lemma 6.3 below rules out more than a failed choice of SWIG for $n = 5$. It also rules out this broader five-bit one-helper framework for the five-player case.

We use the notation and terminology of Definition 6.1. Thus $I_n = \{2, \ldots, n\}$ is read cyclically, and the matrices $T_2, \ldots, T_n$ are invertible.

Lemma 6.2 (5-agreeable conditions on helpers)**.** *Suppose $(\sigma, (T_2, \ldots, T_n))$ satisfies the conditions of Definition 6.1. Then, for every $i \in I_n$, the helper $\sigma(i)$ is neither i nor one of its cyclic neighbors. That is,*

$$\sigma(i) \notin \{i-1, i, i+1\}$$

where addition and subtraction are cyclic in I_n. Moreover, two indices cannot choose each other as helpers:

$$\sigma^2(i) \neq i$$

Proof. We have to prove four cases: the first three cases are that σ cannot pick itself or one of its cyclic neighbors as helpers, and the fourth case is that two indices cannot be helpers to each other. In all four cases, we will contradict the assumption that all $T_2, \ldots, T_n$ must be invertible.

The self-helper case $\sigma(i) = i$ is immediate. If this is the case, then (C2) gives $T_i^\top \mathbf{e}_2 = T_i^\top \mathbf{e}_1$. Two coordinates of T_i would have the same coefficient vector, thus contradicting the invertibility of T_i,

Now suppose i chooses $i-1$ as its helper, making $\sigma(i) = i-1$. From condition (C2), we get $T_{i-1}^\top \mathbf{e}_2 = T_i^\top \mathbf{e}_1$. From condition (C1) at $i-1$, we also get $T_{i-1}^\top \mathbf{e}_3 = T_i^\top \mathbf{e}_1$. Hence $T_{i-1}^\top \mathbf{e}_2 = T_{i-1}^\top \mathbf{e}_3$, which is again impossible.

The next-neighbor case is where i chooses $i+1$ as its helper, so $\sigma(i) = i+1$. Differently from the previous case, we will use condition (C4) instead of (C2). Specifically, (C4) gives $T_{i+1}^\top \mathbf{e}_5 = T_i^\top \mathbf{e}_3$, while (C1) at i gives $T_i^\top \mathbf{e}_3 = T_{i+1}^\top \mathbf{e}_1$. Therefore $T_{i+1}^\top \mathbf{e}_5 = T_{i+1}^\top \mathbf{e}_1$. This collision contradicts the invertibility of T_{i+1}.

It remains to rule out mutual referrals, where both i and $\sigma(i)$ are helpers of each other. That is, $\sigma^2(i) = i$. Applying (C3) at $\sigma(i)$ gives $T_i^\top \mathbf{e}_4 = T_{\sigma(i)}^\top \mathbf{e}_2$. Applying (C2) at i gives $T_{\sigma(i)}^\top \mathbf{e}_2 = T_i^\top \mathbf{e}_1$. Thus $T_i^\top \mathbf{e}_4 = T_i^\top \mathbf{e}_1$, a contradiction again.

Thus, every forbidden helper choice is impossible, as it would violate the invertibility of (at least one of) the matrices $T_2, \ldots, T_n$. □

Lemma 6.3. *The integer $n = 5$ is not 5-agreeable.*

Proof. Consider any index $i \in I_5$. Lemma 6.2 forbids four assignments to the helper role $\sigma(i)$. Specifically, it rules out the following helper choices: i itself, the previous cyclic neighbor $i-1$, and the next cyclic neighbor $i+1$ and mutual helpers.

Now, observe that since the cycle has only four elements in I_5, there is only one index left which is not forbidden, namely $i+2$. We can conclude that for any i, there is only one choice left for $\sigma(i)$, namely $\sigma(i) = i+2$. However, this map has order 2: applying it twice returns every index to its starting position in I_5. Hence, $\sigma^2(i) = i$ for every $i \in I_5$ which contradicts the last forbidden condition of lemma 6.2. We conclude that no admissible helper permutation exists for I_5, and $n = 5$ is not 5-agreeable. □

Remark 6.1. Lemma 6.3 rules out this specific five-bit one-helper recycling framework for $n = 5$, even with an arbitrary helper permutation. It does not prove that $\beta_5 > 5$, and the value of β_5 is still open. Couteau and Rosén give a six-bit no-helper AND protocol for every $n \geq 3$. In particular, we know that $\beta_5 \leq 6$ [CR22, Theorem 14].

6.2 Proof of Theorem 1.3

Proof of Theorem 1.3. For every $n \geq 6$, Theorem 1.5 gives a $(5, n-1)$-SWIG. Our Lemma 4.2 applies protocol in Figure 7 with helper map $\sigma(i) = i - 2$. The resulting protocol is correct, perfectly 1-private, and uses exactly the five-bit seed sampled by $\mathcal{P}_1$. Thus $\beta_n \leq 5$ for all $n \geq 6$. □

Acknowledgments

Supported in part by NSF grants CNS-2246355 and CCF-2220450, US-Israel BSF grant 2022370, and by Sunday Group.

References

[CKOR97] Ran Canetti, Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosén. Randomness vs. fault-tolerance. In James E. Burns and Hagit Attiya, editors, *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997*, pages 35–44. ACM, 1997.

[CKOR00] Ran Canetti, Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosen. Randomness versus fault-tolerance. *Journal of Cryptology*, 13(1):107–142, 2000.

[CR22] Geoffroy Couteau and Adi Rosén. Random sources in private computation. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 443–473. Springer, 2022.

[EL86] T. Etzion and A. Lempel. An efficient algorithm for generating linear transformations in a shuffle-exchange network. *SIAM Journal on Computing*, 15(1):216–221, February 1986.

[GIS22] Vipul Goyal, Yuval Ishai, and Yifan Song. Tight bounds on the randomness complexity of secure multiparty computation. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA,*

August 15-18, 2022, Proceedings, Part IV, volume 13510 of *Lecture Notes in Computer Science*, pages 483–513. Springer, 2022.

[GKM11] Anant P. Godbole, Markos V. Koutras, and Fotios S. Milienos. Binary consecutive covering arrays. *Annals of the Institute of Statistical Mathematics*, 63(3):559–584, 2011.

[HL19] Ishay Haviv and Michael Langberg. H-wise Independence. *Chicago Journal of Theoretical Computer Science*, 2019(3), 2019.

[IKL+13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 576–588. Springer, 2013.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KM97] Eyal Kushilevitz and Yishay Mansour. Randomness in private computations. *SIAM J. Discret. Math.*, 10(4):647–661, 1997.

[KOP+19] Eyal Kushilevitz, Rafail Ostrovsky, Emmanuel Prouff, Adi Rosén, Adrian Thillard, and Damien Vergnaud. Lower and upper bounds on the randomness complexity of private computations of AND. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 386–406. Springer, 2019.

[KOP+21] Eyal Kushilevitz, Rafail Ostrovsky, Emmanuel Prouff, Adi Rosén, Adrian Thillard, and Damien Vergnaud. Lower and upper bounds on the randomness complexity of private computations of AND. *SIAM Journal on Discrete Mathematics*, 35(1):465–484, 2021.

[KOR96] Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosén. Characterizing linear size circuits in terms of privacy. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 541–550. ACM, 1996.

[KOR99] Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosén. Characterizing linear size circuits in terms of privacy. *Journal of Computer and System Sciences*, 58(1):129–136, February 1999.

[LT89] Nathan Linial and Michael Tarsi. Interpolation between bases and the shuffle exchange network. *European Journal of Combinatorics*, 10(1):29–39, 1989.

[MRS19] Lucia Moura, Sebastian Raaphorst, and Brett Stevens. Upper bounds on the sizes of variable strength covering arrays using the Lovász local lemma. *Theoretical Computer Science*, 800:146–154, 2019.

[RMS18] Sebastian Raaphorst, Lucia Moura, and Brett Stevens. Variable strength covering arrays. *Journal of Combinatorial Designs*, 26(9):417–438, 2018.

[Sch92] Leonard J. Schulman. Sample spaces uniform on neighborhoods. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 17–25. ACM, 1992.

[SWL16] Ce Shi, Bin Wen, and Jing Lin. Cyclic consecutive orthogonal arrays. *Acta Mathematicae Applicatae Sinica (Chinese Series)*, 39(5):786–800, 2016.