TinyKeys: A New Approach to Efficient Multi-Party Computation

Carmit Hazay[1⋆], Emmanuela Orsini[2⋆⋆], Peter Scholl[3⋆⋆⋆], and Eduardo Soria-Vazquez[4†]

[1] Bar-Ilan University, Israel
carmit.hazay@biu.ac.il
[2] KU Leuven ESAT/COSIC, Belgium
emmanuela.orsini@kuleuven.be
[3] Aarhus University, Denmark
peter.scholl@cs.au.dk
[4] University of Bristol, UK
eduardo.soria-vazquez@bristol.ac.uk

Abstract. We present a new approach to designing concretely efficient MPC protocols with semi-honest security in the dishonest majority setting. Motivated by the fact that within the dishonest majority setting the efficiency of most practical protocols *does not depend on the number of honest parties*, we investigate how to construct protocols which improve in efficiency as the number of honest parties increases. Our central idea is to take a protocol which is secure for $n-1$ corruptions and modify it to use short symmetric keys, with the aim of basing security on the concatenation of all honest parties' keys. This results in a more efficient protocol tolerating fewer corruptions, whilst also introducing an LPN-style syndrome decoding assumption.

We first apply this technique to a modified version of the semi-honest GMW protocol, using OT extension with short keys, to improve the efficiency of standard GMW with fewer corruptions. We also obtain more efficient constant-round MPC, using BMR-style garbled circuits with short keys, and present an implementation of the online phase of this protocol. Our techniques start to improve upon existing protocols when there are around $n = 20$ parties with $h = 6$ honest parties, and as these increase we obtain up to a 13 times reduction (for $n = 400, h = 120$) in communication complexity for our GMW variant, compared with the best-known GMW-based protocol modified to use the same threshold.

⋆ Supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS), and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

⋆⋆ Supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT.

⋆⋆⋆ Supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583 (SODA), and the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC).

† Supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 643161, and by ERC Advanced Grant ERC-2015-AdG-IMPaCT.

Table of Contents

1 Introduction

Secure multi-party computation (MPC) protocols allow a group of n parties to compute some function f on the parties' private inputs, while preserving a number of security properties such as *privacy* and *correctness*. The former property implies data confidentiality, namely, nothing leaks from the protocol execution but the computed output. The latter requirement implies that the protocol enforces the integrity of the computations made by the parties, namely, honest parties are not lead to accept a wrong output. Security is proven either in the presence of an honest-but-curious adversary that follows the protocol specification but tries to learn more than allowed from its view of the protocol, or a malicious adversary that can arbitrarily deviate from the protocol specification in order to compromise the security of the other parties in the protocol.

The efficiency of a protocol typically also depends on how many corrupted parties can be tolerated before security breaks down, a quantity known as the *threshold*, t. With semi-honest security, most protocols either require $t < n/2$ (where n is the number of parties), in which case unconditionally secure protocols [BOGW88, CCD88] based on Shamir secret-sharing can be used, or support any choice of t up to $n-1$, as in computationally secure protocols based on oblivious transfer [GMW87, Gol04]. Interestingly, within these two ranges, the efficiency of most practical semi-honest protocols *does not depend on* t. For instance, the GMW [GMW87] protocol (and its many variants) is *full-threshold*, so supports any $t < n$ corruptions. However, we *do not know* of any practical protocols with threshold, say, $t = \frac{2}{3}n$, or even $t = n/2 + 1$, that are more efficient than full-threshold GMW-style protocols. One exception to this is when the number of parties becomes very large, in which case protocols based on *committees* can be used. In this approach, due to an idea of Bracha [Bra85], first a random committee of size $n' \ll n$ is chosen. Then every party secret-shares its input to the parties in the committee, who runs a secure computation protocol for $t < n'$ to obtain the result. The committee size n' must be chosen to ensure (with high probability) that not the whole committee is corrupted, so clearly a lower threshold t allows for smaller committees, giving significant efficiency savings. However, this technique is only really useful when n is very large, at least in the hundreds or thousands.

In this paper we investigate designing MPC protocols where *an arbitrary threshold for the number of corrupted parties can be chosen*, which are practical both when n is very large, and also for small to medium sizes of n. Specifically, we ask the question:

> *Can we design* concretely efficient *MPC protocols where the performance improves gracefully as the number of honest parties increases?*

Note that the performance of an MPC protocol can be measured both in terms of *communication overhead* and *computational overhead*. Using fully homomorphic encryption [Gen09], it is possible to achieve very low communication overhead that is independent of the circuit size [AJL+12] even in the malicious setting, but for reasonably complex functions FHE is impractical due to very high computational costs. On the other hand, practical MPC protocols typically communicate for every AND gate in the circuit, and use *oblivious transfer* (OT) to carry out the computation. Fast OT extension techniques allow a large number of secret-shared bit multiplications[5] to be performed using only symmetric primitives and an amortized communication complexity of $O(\kappa)$ [IKNP03] or $O(\kappa/\log\kappa)$ [KK13, DKS+17] bits, where κ is a computational security parameter. This leads to an overall communication complexity which grows with $O(n^2\kappa/\log\kappa)$ bits per AND gate in protocols based on secret-sharing following the [GMW87] style, and $O(n^2\kappa)$ in those based on garbled circuits in the style of [Yao86, BMR90, BLO16].

[5] Note that OT is equivalent to secret-shared bit multiplication, and when constructing MPC it is more convenient to use the latter definition.

Short keys for secure computation. Our main idea towards achieving the above goal is to build a secure multi-party protocol with h honest parties, by distributing secret key material so that each party only holds a *small part of the key*. Instead of basing security on secret keys held by each party individually, we then base security on the *concatenation of all honest parties' keys*.

As a toy example, consider the following simple distributed encryption of a message m under n keys:

$$\mathsf{E}_k(m) = \bigoplus_{i=1}^{n} \mathsf{H}(i, k_i) \oplus m$$

where H is a suitable hash function and each key $k_i \in \{0,1\}^\ell$ belongs to party P_i. In the full-threshold setting with up to $n-1$ corruptions, to hide the message we need each party's key to be of length $\ell = 128$ to achieve 128-bit computational security. However, if only $t < n-1$ parties are corrupted, it seems that, intuitively, an adversary needs to guess all $h := n - t$ honest parties' keys to recover the message, and potentially each key k_i can be *much less than 128 bits* long when h is large enough. This is because the "obvious" way to try to guess m would be to brute force all h keys until decrypting "successfully".

In fact, recovering m when there are h unknown keys corresponds to solving an instance of the *regular syndrome decoding problem* [AFS03], which is related to the well-known *learning parity with noise* (LPN) problem, and believed to be hard for suitable choices of parameters.

1.1 Our Contribution

In this work we use the above idea of short secret keys to design new MPC protocols in both the constant round and non-constant round settings, which improve in efficiency as the number of honest parties increases. We consider security against a static, honest-but-curious adversary, and leave it for future work to extend our techniques to the malicious case based on, e.g. message authentication codes. Our contribution is captured by the following:

GMW-STYLE MPC WITH SHORT KEYS (SECTION 3). We present a GMW-style MPC protocol for binary circuits, where multiplications are done with OT extension using short symmetric keys. This reduces the communication complexity of OT extension-based GMW from $O(n^2\kappa/\log\kappa)$ [KK13] to $O(nt\ell)$, where the key length ℓ decreases as the number of honest parties, $h = n - t$, increases. When h is large enough, we can even have ℓ *as small as* 1.
To construct this protocol, we first analyse the security of the IKNP OT extension protocol [IKNP03] when using short keys, and formalise the leakage obtained by a corrupt receiver in this case. We then show how to use this version of "leaky OT" to generate multiplication triples using a modified version of the GMW method, where pairs of parties use OT to multiply their shares of random values. We also optimize our protocol by reducing the number of communication channels using two different-sized committees, improving upon the standard approach of choosing one committee to do all the work.

MULTI-PARTY GARBLED CIRCUITS WITH SHORT KEYS (SECTION 4). Our second contribution is the design of a constant round, BMR-style [BMR90] protocol based on garbled circuits with short keys. Our offline phase uses the multiplication protocol from the previous result in order to generate the garbled circuit, using secret-shared bit and bit/string multiplications as done in previous works [BLO16, HSS17], with the exception that the keys are shorter. In the online phase, we then use the LPN-style assumption to show that the combination of all honest parties' ℓ-bit keys suffices to obtain a secure garbling protocol. This allows us to save on the key length as a function of the number of honest parties.

As well as reducing communication with a smaller garbled circuit, we also reduce computation when evaluating the circuit, since each garbled gate can be evaluated with only $O(n^2\ell/\kappa)$ block cipher calls (assuming the ideal cipher model), instead of $O(n^2)$ when using κ-bit keys. For this protocol, ℓ *can be as small as 5* when n is large enough, giving a significant saving over 128-bit keys used previously.

Concrete Efficiency Improvements. The efficiency of our protocols depends on the total number of parties, n, and the number of honest parties, h, so there is a large range of parameters to explore when comparing with other works. We discuss this in more detail in Section 5. Our protocols seem most significant in the *dishonest majority* setting, since when there is an honest majority there are unconditionally secure protocols with $O(n \log n)$ communication overhead and reasonable computational complexity e.g. [DN07], whilst our protocols have $\Omega(nt)$ communication overhead.

Our GMW-style protocol starts to improve upon previous protocols when we reach $n = 20$ parties and $t = 14$ corruptions: here, our triple generation method requires less than *half the communication cost* of the fastest GMW-style protocol based on OT extension [DKS+17] tolerating up to $n - 1$ corruptions. When the number of honest parties is large enough, we can use *1-bit keys*, giving a *25-fold reduction* in communication over previous protocols when $n = 400$ and $t = 280$. In addition, we describe a simple threshold-t variant of GMW-style protocols, which our protocol still outperforms by 1.1x and 13x, respectively, in these two scenarios.

For our constant round protocol, with $n = 20, t = 10$ we can use 32-bit keys, so the size of each garbled AND gate is $1/4$ the size of [BLO16]. As n increases the improvements become greater, with a *16-fold reduction* in garbled AND gate size for $n = 400, t = 280$. We also reduce the communication cost of *creating* the garbled circuit. Here, the improvement starts at around 50 parties, and goes up to a 7 times reduction in communication when $n = 400, t = 280$. Note that our protocol does incur a slight additional overhead, since we need to use extra "splitter gates", but this cost is relatively small.

To demonstrate the practicality of our approach, we also present an implementation of the online evaluation phase of our constant-round protocol for key lengths ranging between $1 - 4$ bytes, and with an overall number of parties ranging from $15 - 1000$; more details can be found in Section 5.

Applications. Our techniques seem most useful for large-scale MPC with around 70% corruptions, where we obtain the greatest concrete efficiency improvements. An important motivation for this setting is privacy-preserving statistical analysis of data collected from a large network with potentially thousands of nodes. In scenarios where the nodes are not always online and connected, our protocols can also be used with the "random committee" approach discussed earlier, so only a small subset of, say, a hundred nodes need to be online and interacting during the protocol.

An interesting example is safely measuring the Tor network [DMS04] which is among the most popular tools for digital privacy, consisting of more than 6000 relays that can opt-in for providing statistics about the use of the network. Nowadays and due to privacy risks, the statistics collected over Tor are generally poor: There is a reduced list of computed functions and only a minority of the relays provide data, which has to be obfuscated before publishing [DMS04]. Hence, the statistics provide an incomplete picture which is affected by a noise that scales with the number of relays. Running MPC in this setting would enable for more complex, accurate and private data processing, for example through anomaly detection and more sophisticated censorship detection. Moreover, our protocols are particularly well-suited to this setting since all relays in the network must be connected to one another already, by design.

Another possible application is for securely computing the interdomain routing within the Border Gateway Protocol (BGP), which is performed at a large scale of thousands of nodes. A recent solution in the

dishonest majority setting [ADS+17] centralizes BGP so that two parties run this computation for all Autonomous Systems. Our techniques allow scaling to a large number of systems computing the interdomain routing themselves using MPC, hence further reducing the trust requirements.

Decisional Regular Syndrome Decoding problem. The security of our protocols relies on the *Decisional Regular Syndrome Decoding (DRSD)* problem, which, given a random binary matrix $\mathbf{H}$, is to distinguish between the syndrome obtained by multiplying $\mathbf{H}$ with an error vector $\boldsymbol{e} = (\boldsymbol{e}_1 \| \cdots \| \boldsymbol{e}_h)$ where each $\boldsymbol{e}_i \in \{0,1\}^{2^\ell}$ has Hamming weight one, and the uniform distribution. This can equivalently be described as distinguishing $\bigoplus_{i=1}^{h} \mathsf{H}(i, k_i)$ from the uniform distribution, where H is a random function and each k_i is a random ℓ-bit key (as in the toy example described earlier).

We remark that when h is large enough, the problem is *unconditionally hard* even for $\ell = 1$, which means for certain parameter choices in our GMW-based protocol we can use 1-bit keys *without introducing any additional assumptions*. This introduces a significant saving in our triple generation protocol.

Overall, our approach demonstrates a new application of LPN-type assumptions to efficient MPC without introducing asymmetric operations. Our techniques may also be useful in other distributed applications where only a small fraction of nodes are honest.

Additional related work. Another work which applies a similar assumption to secure computation is that of Applebaum [App16], who built garbled circuits with the free-XOR technique in the standard model under the LPN assumption. Conceptually, our work differs from Applebaum's since our focus is to improve the efficiency of multi-party protocols with fewer corruptions, whereas in [App16], LPN is used in a more modular way in order to achieve encryption with stronger properties and under a more standard assumption.

In a recent work [NR17], Nielsen and Ranellucci designed a protocol in the dishonest majority setting with malicious, adaptive security in the presence of $t < cn$ corruption for $t \in [0, 1)$. Their protocol is aimed to work with a large number of parties and uses committees to obtain a protocol with poly-logarithmic overhead. This protocol introduces high constants and is not useful for practical applications.

Finally, in a concurrent work [BO17], Ben-Efraim and Omri also explore how to optimize garbled circuits in the presence of non-full-threshold adversaries. By using deterministic committees they achieve AND gates of size $4(t+1)\kappa$, where κ is the computational security parameter. By using the same technique we achieve a size of $4(t+h)\ell$, where $\ell \ll \kappa$ depends on h, a parameter for the minimum number of honest parties in the committee. The rest of their results apply only to the honest majority setting.

1.2 Technical Overview

In what follows we explain the technical side of our results in more detail.

Leaky oblivious transfer (OT). We first present a two-party secret-shared bit multiplication protocol, based on a variant of the IKNP OT extension protocol [IKNP03] with short keys. Our protocol performs a batch of r multiplications at once. Namely, the parties create r correlated OTs on ℓ-bit strings using the OT extension technique of [IKNP03], by transposing a matrix of ℓ OTs on r-bit strings and swapping the roles of sender and receiver. In contrast to the IKNP OT extension and followups, that use κ 'base' OTs for computational security parameter κ, we use $\ell = O(\log \kappa)$ base OTs.

This protocol leaks some information on the global secret $\Delta \leftarrow \{0,1\}^\ell$ picked by the receiver, as well as the inputs of the receiver. Roughly speaking, the leakage is of the form $\mathsf{H}(i, \Delta) + x_i$, where $x_i \in \{0,1\}$ is an input of the receiver and H is a hash function with 1-bit output. Clearly, when ℓ is short this is not secure to use on its own, since all of the receiver's inputs only have ℓ bits of min-entropy (based on the choice of Δ).

MPC from leaky OT. We then show how to apply this leaky two-party protocol to the multi-party setting, whilst preventing any leakage on the parties shares. The main observation is that, when using additive secret-sharing, we only need to ensure that the *sum* of all honest parties' shares is unpredictable; if the adversary learns just a few shares, they can easily be rerandomized by adding pseudorandom shares of zero, which can be done non-interactively using a PRF. However, we still have a problem, which is that in the standard GMW approach, each party P_i uses OT to multiply their share x^i with every other party P_j's share y^j. Now, there is leakage on the *same share* x^i from each of the OT instances between all other parties, which seems much harder to prevent than leakage from just a single OT instance.

To work around this problem, we have the parties add shares of zero to their x^i inputs *before* multiplying them. So, every pair (P_i, P_j) will use leaky OT to multiply $x^i \oplus s^{i,j}$ with y^j, where $s^{i,j}$ is a random share of zero satisfying $\bigoplus_{i=1}^n s^{i,j} = 0$. This preserves correctness of the protocol, because the parties end up computing an additive sharing of:

$$\bigoplus_{i=1}^{n}\bigoplus_{j=1}^{n}(x^i \oplus s^{i,j})y^j = \bigoplus_{j=1}^{n} y^j \bigoplus_{i=1}^{n}(x^i \oplus s^{i,j}) = xy.$$

This also effectively removes leakage on the individual shares, so we only need to be concerned with the *sum* of the leakage on all honest parties' shares, and this turns out to be of the form: $\bigoplus_{i=1}^n(\mathsf{H}(i, \Delta_i) + x^i)$ which is pseudorandom under the decisional regular syndrome decoding assumption.

We realize our protocol using a hash function with a polynomial-sized domain, so that is can be implemented using a CRS which simply outputs a random lookup-table. This means that, unlike when using the IKNP protocol, we do not need to rely on a random oracle or a correlation robustness assumption.

When the number of parties is large enough, we can improve our triple generation protocol using *random committees*. In this case the amortized communication cost is $\leq n_h n_1(\ell + \ell\kappa/r + 1)$ bits per multiplication where we need to choose two committees of sizes n_h and n_1 which have at least h and 1 honest parties, respectively.

Garbled circuits with short keys. We next revisit the multi-party garbled circuits technique by Beaver, Micali and Rogaway, known as BMR, that extends the classic Yao garbling [Yao86] to an arbitrary number of parties, where essentially all the parties jointly garble using one set of keys each. This method was recently improved in a sequence of works [LPSY15, LSS16, BLO16, HSS17], where the two latter works further support the Free-XOR property.

Our garbling method uses an expansion function $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}^{n\ell+1}$, where ℓ is the length of each parties' keys used as wire labels in the garbled circuit. To garble a gate, the hash values of the input wire keys $k^i_{u,b}$ and $k^i_{v,b}$ are XORed over i and used to mask the output wire keys.

Specifically, for an AND gate g with input wires u, v and output wire w, the 4 garbled rows $\tilde{g}_{a,b}$, for each $(a, b) \in \{0,1\}^2$, are computed as:

$$\tilde{g}_{a,b} = \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, b, k^i_{u,a}) \oplus \mathsf{H}(i, a, k^i_{v,b})\right) \oplus (c, k^1_{w,c}, \ldots, k^n_{w,c}).$$

Security then relies on the DRSD assumption, which implies that the sum of h hash values on short keys is pseudorandom, which suffices to construct a secure garbling method with h honest parties.

Using this assumption instead of a PRF (as in recent works) comes with difficulties, as we can no longer garble gates with arbitrary fan-out, or use the free-XOR technique, without degrading the DRSD parameters. To allow for arbitrary fan-out circuits with our protocol we use *splitter gates*, which take as input one wire

w and provide two outputs wires u, v, representing the same wire value. Splitter gates were previously introduced as a fix for an error in the original BMR paper in [TX03]. We stress that transforming a general circuit description into a circuit with only fan-out-1 gates requires adding at most a single splitter gate per AND or XOR gate.

The restriction to fan-out-1 gates and the use of splitter gates additionally allows us to garble XOR gates for free in BMR without relying on circular security assumptions or correlation-robust hash functions, based on the FlexOR technique [KMR14] where each XOR gate uses a unique offset. Furthermore, the overhead of splitter gates is very low, since garbling a splitter gate does not use the underlying MPC protocol: shares of the garbled gate can be generated non-interactively. We note that this observation also applies to Yao's garbled circuits, but the overhead of adding splitter gates there is more significant; this is because in most 2-party protocols, the *size* of the garbled circuit is the dominant cost factor, whereas in multi-party protocols the main cost is *creating* the garbled circuit in a distributed manner.

2 Preliminaries

We denote the security parameter by κ. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. The function μ is *noticeable* (or non-negligible) if there exists a positive polynomial $p(\cdot)$ such that for all sufficiently large κ it holds that $\mu(\kappa) \geq \frac{1}{p(\kappa)}$. We use the abbreviation PPT to denote probabilistic polynomial-time. We further denote by $a \leftarrow A$ the uniform sampling of a from a set A, and by $[d]$ the set of elements $\{1, \ldots, d\}$. We often view bit-strings in $\{0,1\}^k$ as vectors in $\mathbb{F}_2^k$, depending on the context, and denote exclusive-or by "$\oplus$" or "$+$". If $a, b \in \mathbb{F}_2$ then $a \cdot b$ denotes multiplication (or AND), and if $\boldsymbol{c} \in \mathbb{F}_2^\kappa$ then $a \cdot \boldsymbol{c} \in \mathbb{F}_2^\kappa$ denotes the product of a with every component of $\boldsymbol{c}$.

We first specify the definition of computational indistinguishability.

Definition 2.1 *Let $X = \{X(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ be two distribution ensembles. We say that X and Y are* computationally indistinguishable*, denoted $X \overset{c}{\approx} Y$, if for every* PPT *machine $\mathcal{D}$ and every $a \in \{0,1\}^*$, there exists a negligible function* negl *such that:*

$$\left|\Pr\left[\mathcal{D}(X(a,\kappa), a, 1^\kappa) = 1\right] - \Pr\left[\mathcal{D}(Y(a,\kappa), a, 1^\kappa) = 1\right]\right| < \mathsf{negl}(\kappa).$$

2.1 Security and Communication Models

We prove security of our protocols in the universal composability (UC) framework [Can01]. See Appendix A for a summary of this. We assume all parties are connected via secure, authenticated point-to-point channels, which is the default method of communication in our protocols. The adversary model we consider is a static, honest-but-curious adversary who corrupts a subset $A \subset [n]$ of parties at the beginning of the protocol. We denote by $\bar{A}$ the subset of honest parties, and define $h = |\bar{A}| = n - t$.

2.2 Random Zero-Sharing

Our protocols require the parties to generate random additive sharings of zero, as in the $\mathcal{F}_{\text{Zero}}$ functionality in Figure 1. This can be done efficiently using a PRF F, with interaction *only* during a setup phase, as in [AFL+16]. We do this by asking each party P_i to send a random PRF key $k_{i,j}$ to every other party P_j. Next, P_i defines its share by $\bigoplus_{j\neq i}(F_{k_{i,j}}(\tau) \oplus F_{k_{j,i}}(\tau))$ where τ is an index that identifies the generated share. It is simple to verify that all the shares XOR to zero since each PRF value is used exactly twice.

Functionality $\mathcal{F}^r_{\mathrm{Zero}}(\mathcal{P})$

On receiving $(\texttt{zero})$ from all parties in $\mathcal{P} = \{P_1, \ldots, P_n\}$:

1. Sample random shares $\boldsymbol{s}^2, \ldots, \boldsymbol{s}^n \leftarrow \{0,1\}^r$ and let $\boldsymbol{s}^1 = \boldsymbol{s}^2 \oplus \cdots \oplus \boldsymbol{s}^n$
2. Send $\boldsymbol{s}^i$ to party P_i

Fig. 1. Random zero sharing functionality.

Moreover, privacy holds in the presence of any subset of $n-2$ corrupted parties because the respective values $F_{k_{l,l'}}$ and $F_{k_{l',l}}$ of honest parties P_l and $P_{l'}$ are pseudorandom, which implies that their zero shares are also pseudorandom. Finally, the communication complexity of the setup phase amounts to sending $O(n^2)$ PRF keys, whilst creating the shares requires $2(n-1)$ PRF evaluations to produce κ bits.

2.3 Regular Syndrome Decoding Problem

We now describe the Regular Syndrome Decoding (RSD) problem and some of its properties.

Definition 2.2 *A vector $\boldsymbol{e} \in \mathbb{F}_2^m$ is* (m,h)-regular *if $\boldsymbol{e} = (\boldsymbol{e}_1 \| \cdots \| \boldsymbol{e}_h)$ where each $\boldsymbol{e}_i \in \{0,1\}^{m/h}$ has Hamming weight one. We denote by $R_{m,h}$ the set of all the (m,h)-regular vectors in $\mathbb{F}_2^m$.*

Definition 2.3 (Regular Syndrome Decoding (RSD)) *Let $r, h, \ell \in \mathbb{N}$ with $m = h \cdot 2^\ell$, $\mathbf{H} \leftarrow \mathbb{F}_2^{r \times m}$ and $\boldsymbol{e} \leftarrow R_{m,h}$. Given $(\mathbf{H}, \mathbf{H}\boldsymbol{e})$, the $\mathsf{RSD}_{r,h,\ell}$ problem is to recover $\boldsymbol{e}$ with noticeable probability.*

The decisional version of the problem, given below, is to distinguish the syndrome $\mathbf{H}\boldsymbol{e}$ from uniform.

Definition 2.4 (Decisional Regular Syndrome Decoding (DRSD)) *Let $\mathbf{H} \leftarrow \mathbb{F}_2^{r \times m}$ and $\boldsymbol{e} \leftarrow R_{m,h}$, and let U_r be the uniform distribution on r bits. The $\mathsf{DRSD}_{r,h,\ell}$ problem is to distinguish between $(\mathbf{H}, \mathbf{H}\boldsymbol{e})$ and $(\mathbf{H}, U_r)$ with noticeable advantage.*

Hash function formulation. The DRSD problem can be equivalently described as distinguishing from uniform $\bigoplus_{i=1}^h \mathsf{H}(i, k_i)$ where $\mathsf{H} : [h] \times \{0,1\}^\ell \to \{0,1\}^r$ is a random hash function, and each $k_i \leftarrow \{0,1\}^\ell$. With this formulation, it is easier to see how the DRSD problem arises when using our protocols with short keys, since this appears when summing up a hash function applied to h honest parties' secret keys.

To see the equivalence, we can define a matrix $\mathbf{H} \in \mathbb{F}_2^{r \times h \cdot 2^\ell}$, where for each $i \in \{0, \ldots, h-1\}$ and $k \in [2^\ell]$, column $i \cdot 2^\ell + k$ of $\mathbf{H}$ contains $\mathsf{H}(i,k)$. Then, multiplying $\mathbf{H}$ with a random (m,h)-regular vector $\boldsymbol{e}$ is equivalent to taking the sum of H over h random inputs, as above.

Statistical hardness of DRSD. We next observe that for certain parameters where the output size of $\mathbf{H}$ is sufficiently smaller than the min-entropy of the error vector $\boldsymbol{e}$, the distribution in the decisional problem is *statistically close to uniform*.

Lemma 2.1 *If $\ell = 1$ and $h \geq r + s$ then $\mathsf{DRSD}_{r,h,\ell}$ is statistically hard, with distinguishing probability 2^{-s}.*

Proof. Suppose $\ell = 1$ and $h \geq r + s$, so $m = 2h$. For a vector $\boldsymbol{e} = (\boldsymbol{e}_1 \| \cdots \| \boldsymbol{e}_h) \in R_{m,h}$, we can write each of the weight-1 vectors $\boldsymbol{e}_i \in \{0,1\}^2$ as $(e'_i, 1 - e'_i)$. An RSD sample $\mathbf{H}, \boldsymbol{y} = \mathbf{H}\boldsymbol{e}$ therefore defines a system of r linear equations in the h variables $\{e'_i\}_i$, and it can be shown that this simplifies to the form $\boldsymbol{y} = \mathbf{H}'\boldsymbol{e}' + \boldsymbol{c}$, where $\boldsymbol{e}' = (e'_1, \ldots, e'_h)$, by defining the j-th column of $\mathbf{H}' \in \mathbb{F}_2^{r \times h}$ to be the sum of columns

$2j-1$ and $2j$ from $\mathbf{H}$, and $\boldsymbol{c}$ to be the sum of all even-indexed columns in $\mathbf{H}$. Note that $\mathbf{H}'$ is uniformly random because $\mathbf{H}$ is, and it is easy to show (e.g. [Pie12, Lemma 1]) that the probability that $\mathbf{H}' \leftarrow \mathbb{F}_2^{r\times h}$ is not full rank is no more than 2^{-s} when $h \geq r+s$. Assuming that $\mathbf{H}'$ has full rank and $h \geq r$, $\boldsymbol{y} = \mathbf{H}'\boldsymbol{e}' + \boldsymbol{c}$ must be uniformly random because $\boldsymbol{e}'$ is. □

For the general case of ℓ-bit keys, we use the following form of the leftover hash lemma.

Lemma 2.2 (Leftover Hash Lemma [ILL89]) *Let $\mathbf{H} \leftarrow \mathbb{F}_2^{r\times m}$ and $\boldsymbol{e} \leftarrow \chi$, where χ is a distribution over $\mathbb{F}_2^m$ with min-entropy at least k. If $r \leq k - 2s$ then*

$$\Delta_{SD}((\mathbf{H}, \mathbf{H}\boldsymbol{e}), (\mathbf{H}, \boldsymbol{u})) \leq 2^{-s}$$

where $\boldsymbol{u} \leftarrow \mathbb{F}_2^r$ and Δ_{SD} is the statistical distance.

Note that if $\boldsymbol{e} \leftarrow R_{m,h}$ then we have $H_\infty(\boldsymbol{e}) = h\ell$. Applying Lemma 2.2 with $k = h\ell$, we obtain the following.

Corollary 2.1. *If $h \geq (r+2s)/\ell$ then $\mathsf{DRSD}_{r,h,\ell}$ is statistically hard, with distinguishing probability 2^{-s}.*

Search-to-decision reduction. For *all* parameter choices of DRSD, there is a simple reduction to the search version of the regular syndrome decoding problem with the same parameters.

Lemma 2.3 *Any efficient distinguisher for the $\mathsf{DRSD}_{r,h,\ell}$ problem can be used to efficiently solve $\mathsf{RSD}_{r,h,\ell}$.*

The proof (inspired by a similar result for LPN [AIK09]) is a simplified version of previous reductions for syndrome decoding.

We first recall the Goldreich-Levin hardcore-bit theorem.

Theorem 2.5 ([GL89]) *Let f be a one-way function. Then, given $(r, f(x))$ for uniformly random r and x, the inner product $\langle x, r\rangle$ over $\mathbb{F}_2$ is unpredictable.*

Proof. (of Lemma 2.3) Suppose $\mathcal{A}$ distinguishes between $(\mathbf{H}, \mathbf{H}\boldsymbol{e})$ and $(\mathbf{H}, U_r)$ with noticeable advantage δ. We construct an adversary $\mathcal{A}'$ that breaks the Goldreich-Levin hardcore bit of $f(\boldsymbol{e}) = (\mathbf{H}, \mathbf{H}\boldsymbol{e})$ by guessing the inner product $\langle \boldsymbol{e}, \boldsymbol{s}\rangle$ for some vector $\boldsymbol{s} \in \mathbb{F}_2^m$. On input $(\mathbf{H}, \boldsymbol{y} = \mathbf{H}\boldsymbol{e}, \boldsymbol{s})$, algorithm $\mathcal{A}'$ proceeds as follows:

1. Sample $\boldsymbol{t} \leftarrow \{0,1\}^r$
2. Compute $\mathbf{H}' = \mathbf{H} - \boldsymbol{t} \cdot \boldsymbol{s}^\top$
3. Run $\mathcal{A}$ on input $(\mathbf{H}', \boldsymbol{y})$
4. Output the same as $\mathcal{A}$

First notice that because $\mathbf{H}$ is uniformly random, $\mathbf{H}'$ is also. Secondly, $\boldsymbol{y} = \mathbf{H}\boldsymbol{e} = (\mathbf{H}' + \boldsymbol{t}\cdot\boldsymbol{s}^\top)\boldsymbol{e} = \mathbf{H}'\boldsymbol{e} + \boldsymbol{t}\cdot\langle \boldsymbol{s}, \boldsymbol{e}\rangle$. So, if $\langle \boldsymbol{s}, \boldsymbol{e}\rangle = 0$ then the input to $\mathcal{A}$ is a correct sample $(\mathbf{H}', \mathbf{H}'\boldsymbol{e})$, whereas if $\langle \boldsymbol{s}, \boldsymbol{e}\rangle = 1$ then the input is uniformly random. Therefore, it holds that:

$$\begin{aligned}
\Pr[\mathcal{A}'(\mathbf{H}, \mathbf{H}\boldsymbol{e}, \boldsymbol{r}) = \langle \boldsymbol{e}, \boldsymbol{r}\rangle] &= \Pr[\mathcal{A}'(\mathbf{H}, \mathbf{H}\boldsymbol{e}, \boldsymbol{r}) = 0 | \langle \boldsymbol{e}, \boldsymbol{r}\rangle = 0] \cdot \Pr[\langle \boldsymbol{e}, \boldsymbol{r}\rangle = 0] + \\
&\quad \Pr[\mathcal{A}'(\mathbf{H}, \mathbf{H}\boldsymbol{e}, \boldsymbol{r}) = 1 | \langle \boldsymbol{e}, \boldsymbol{r}\rangle = 1] \cdot \Pr[\langle \boldsymbol{e}, \boldsymbol{r}\rangle = 1] \\
&= \frac{1}{2} \cdot (\Pr[\mathcal{A}(\mathbf{H}', \mathbf{H}'\boldsymbol{e}) = 0] + (1 - \Pr[\mathcal{A}(\mathbf{H}', U_r) = 0])) \\
&\geq \frac{1}{2} + \frac{\delta}{2} \,.
\end{aligned}$$

□

Multi-Secret RSD. We now consider a variant of DRSD with multiple sets of secrets, where the matrix $\mathbf{H}$ is fixed for each sample. We then reduce this to the standard DRSD problem with the same parameters, with a security loss of the number of secrets.

Definition 2.6 (Multi-Secret DRSD) *Let $\mathbf{H} \leftarrow \mathbb{F}_2^{r\times m}$ and $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_q \leftarrow R_{m,h}$ (as in Definition 2.3). The q-$\mathsf{DRSD}_{r,h,\ell}$ problem is to distinguish between a tuple $(\mathbf{H}, \mathbf{H}\boldsymbol{e}_1, \ldots, \mathbf{H}\boldsymbol{e}_q)$ and $(\mathbf{H}, U_r^q)$ with noticeable advantage.*

Lemma 2.4 *q-$\mathsf{DRSD}_{r,h,\ell}$ is reducible to $\mathsf{DRSD}_{r,h,\ell}$, where the reduction loses a tightness factor of q.*

Proof. The proof is based on a standard hybrid argument with a sequence of $q+1$ hybrid distributions, where each pair of neighbouring hybrids is indistinguishable based on DRSD.
The first hybrid, H_0, outputs $(\mathbf{H}, u_1, \ldots, u_q)$, where $\mathbf{H} \leftarrow \mathbb{F}_2^{r\times m}$ and $u_i \leftarrow \{0,1\}^r$, which is exactly the uniform distribution used in q-DRSD. In hybrid H_i, for $i = 1, \ldots, q$, we sample regular secrets $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_i$ and output $(\mathbf{H}, \mathbf{H}\boldsymbol{e}_1, \ldots, \mathbf{H}\boldsymbol{e}_i, u_{i+1}, \ldots, u_q)$. Note that H_q is the same as the real distribution in the q-DRSD problem. Any adversary $\mathcal{A}$ who distinguishes between H_i and H_{i+1} can be used to break $\mathsf{DRSD}_{r,h,\ell}$, as follows. The distinguisher $\mathcal{D}$ receives a DRSD challenge $(\mathbf{H}, \boldsymbol{y})$, then samples $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_i$ from the error distribution and random strings $u_{i+2}, \ldots, u_q \leftarrow \{0,1\}^r$. It then outputs $\mathcal{A}(\mathbf{H}, \mathbf{H}\boldsymbol{e}_1, \ldots, \mathbf{H}\boldsymbol{e}_i, \boldsymbol{y}, u_{i+2}, \ldots, u_q)$. The advantage of $\mathcal{D}$ against the DRSD problem is identical to that of $\mathcal{A}$. A standard argument then implies that any adversary who distinguishes H_0 and H_q with advantage δ can solve $\mathsf{DRSD}_{r,h,\ell}$ with advantage at least δ/q. □

Extended Double-Key RSD. In our final variant of RSD — used in the security proof of our BMR-style online phase — we consider multiple sets of secrets, and also give the adversary two challenges for each secret which captures the double use of each key in the garbling procedure. This means we cannot preserve the RSD parameters, and must reduce to 2-$\mathsf{DRSD}_{2r,h,\ell}$. We also make a conceptual change, and specify the problem using a random hash function H with small domain (which can be modelled as a random oracle, or a random lookup table generated in a trusted setup phase which can be modelled as a common random string) instead of matrices and vectors. We switch to this notation in order to capture the computation made by the honest parties when garbling a gate.

Definition 2.7 (Extended Double-Key DRSD) *The extended double-key decisional-RSD problem states that, for every fixed subset $S \subset [n]$ of size h, it holds that*

$$\left(\mathsf{H}, \bigoplus_{i\in S}\mathsf{H}(i,0,k_i), \bigoplus_{i\in S}\mathsf{H}(i,0,k_i'), \bigoplus_{i\in S}\mathsf{H}(i,1,k_i), \bigoplus_{i\in S}\mathsf{H}(i,1,k_i')\right) \quad \overset{\mathrm{c}}{\approx} \quad (\mathsf{H}, U_{4r}),$$

where $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}^r$ is a randomly sampled function, and $k_i, k_i' \leftarrow \{0,1\}^\ell$ for $i \in S$.

Lemma 2.5 *The extended double-key decisional-RSD problem with parameters (r, h, ℓ) is reducible to 2-$\mathsf{DRSD}(2r, h, \ell)$.*

Proof. Suppose there exists a set $S \subset [n]$ for which an adversary $\mathcal{A}$ distinguishes the above two distributions with noticeable advantage. We use $\mathcal{A}$ to construct a distinguisher $\mathcal{D}$ for the 2-$\mathsf{DRSD}(2r, h, \ell)$ problem. $\mathcal{D}$ receives a challenge $(\mathbf{H}, \boldsymbol{y}_0, \boldsymbol{y}_1)$, where $\mathbf{H} \in \mathbb{F}_2^{2r\times m}$, $m = h \cdot 2^\ell$ and $\boldsymbol{y}_0, \boldsymbol{y}_1 \in \mathbb{F}_2^{2r}$. Write $\mathbf{H} = \begin{pmatrix}\mathbf{H}_0\\ \mathbf{H}_1\end{pmatrix}$ and $\boldsymbol{y}_j = \begin{pmatrix}\boldsymbol{z}_j\\ \boldsymbol{z}_j'\end{pmatrix}$. Define the hash function $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}^r$ so that $\mathsf{H}(s_i, b, k)$ is equal to

column $2^\ell i + k$ (viewing k also as an integer in $[2^\ell]$) of the matrix $\mathbf{H}_b$, for each $s_i \in S$ and $b \in \{0,1\}$. For $i \in [n] \setminus S$, let the output of $\mathsf{H}(i,\cdot,\cdot)$ be uniformly random. The distinguisher then runs $\mathcal{A}$ with input

$$\left(\mathsf{H}, \boldsymbol{z}_0, \boldsymbol{z}'_0, \boldsymbol{z}_1, \boldsymbol{z}'_1\right),$$

and outputs the same as $\mathcal{A}$. Notice that if the DRSD challenge is random then the input to $\mathcal{A}$ is random, whereas if the challenge is computed as $\boldsymbol{y}_j = \mathbf{H}\boldsymbol{e}_j$ for some regular error $\boldsymbol{e}_j$ and $j \in \{0,1\}$, then we have $\boldsymbol{z}_j = \mathbf{H}_0\boldsymbol{e}_j$ and $\boldsymbol{z}'_j = \mathbf{H}_1\boldsymbol{e}_j$, and by the definition of H, these values are equal to the sum of hash function outputs under some secret keys corresponding to $\boldsymbol{e}_j$. It follows that the distinguishing advantage of $\mathcal{D}$ is the same as that of $\mathcal{A}$. □

3 GMW-Style MPC with Short Keys

In this section we design a protocol for generating multiplication triples over $\mathbb{F}_2$ using short secret keys, with reduced communication complexity as the number of honest parties increases. More concretely, we first design a leaky protocol for secret-shared two-party bit multiplication, based on correlated OT and OT extension techniques with short keys. This protocol is not fully secure and we precisely define the leakage obtained by the receiver. We next show how to use the leaky protocol to produce multiplication triples, removing the leakage by rerandomizing the parties' shares with shares of zero, and using the DRSD assumption. Finally, this protocol can be used with Beaver's multiplication triple technique [Bea92] to obtain MPC for binary circuits with an amortized communication complexity of $O(nt\ell)$ bits per triple, where t is the threshold and ℓ is the secret key length. When the number of honest parties is large enough we can even use $\ell = 1$ and avoid relying on DRSD.

3.1 Leaky Two-Party Secret-Shared Multiplication

Functionality $\mathcal{F}^{r,\ell}_{\Delta\text{-ROT}}$

After receiving $\Delta \in \{0,1\}^\ell$ from P_S and $(x_1, \ldots, x_r) \in \{0,1\}^r$ from P_R, do the following:

1. Sample $\boldsymbol{q}_i \leftarrow \{0,1\}^\ell$, for $i \in [r]$, and let $\boldsymbol{t}_i = \boldsymbol{q}_i \oplus x_i \cdot \Delta$.
2. Output $\boldsymbol{q}_i$ to P_S and $\boldsymbol{t}_i$ to P_R, for $i \in [r]$.

Fig. 2. Functionality for oblivious transfer on random, correlated strings.

We first present our protocol for two-party secret-shared bit multiplication, based on a variant of the [IKNP03] OT extension protocol, modified to use short keys. With short keys we cannot hope for computational security based on standard symmetric primitives, because an adversary can search every possible key in polynomial time. Our goal, therefore, is to define the precise *leakage* that occurs when using short keys, in order to remove this leakage at a later stage.

OT extension and correlated OT. Recall that the main observation of the IKNP protocol for extending oblivious transfer [IKNP03] is that *correlated OT is symmetric*, so that κ correlated OTs on r-bit strings can be locally converted into r correlated OTs on κ-bit strings. Secondly, a κ-bit correlated OT can be used to obtain an OT on chosen strings with computational security. The first stage of this process is abstracted away by the functionality $\mathcal{F}_{\Delta\text{-ROT}}$ in Figure 2.

Using IKNP to multiply an input bit x_k from the sender, P_A, with an input bit y_k from P_B, the receiver, P_B sends y_k as its choice bit to $\mathcal{F}_{\Delta\text{-ROT}}$ and learns $\boldsymbol{t}_k = \boldsymbol{q}_k \oplus y_k \cdot \Delta$. The sender P_A obtains $\boldsymbol{q}_k$, and then sends

$$d_k = \mathsf{H}(\boldsymbol{q}_k) \oplus \mathsf{H}(\boldsymbol{q}_k \oplus \Delta) \oplus x_k,$$

where H is a 1-bit output hash function. This allows the parties to compute an additive sharing of $x_k \cdot y_k$ as follows: P_A defines the share $\mathsf{H}(\boldsymbol{q}_k)$, and P_B computes $\mathsf{H}(\boldsymbol{t}_k) \oplus y_k \cdot d_k$. This can be repeated many times with the same Δ to perform a large batch of $\mathsf{poly}(\kappa)$ secret-shared multiplications, because the randomness in Δ serves to computationally mask each x with the hash values (under a suitable correlation robustness assumption for H). The downside of this is that for $\Delta \in \{0,1\}^\kappa$, the communication cost is $O(\kappa)$ bits per two-party bit multiplication, to perform the correlated OTs.

Variant with short keys. We adapt this protocol to use short keys by performing the correlated OTs on ℓ-bit strings, instead of κ-bit, for some small key length $\ell = O(\log \kappa)$ (we could have ℓ as small as 1). This allows $\mathcal{F}_{\Delta\text{-ROT}}$ to be implemented with only $O(\ell)$ bits of communication per OT instead of $O(\kappa)$.

Our protocol, shown in Figure 4, performs a batch of r multiplications at once. First the parties create r correlated OTs on ℓ-bit strings using $\mathcal{F}_{\Delta\text{-ROT}}$. Next, the parties hash the output strings of the correlated OTs, and P_A sends over the correction values d_k, which are used by P_B to convert the random OTs into a secret-shared bit multiplication. Finally, we require the parties to add a random value (from $\mathcal{F}_{\text{Zero}}$, shown in Figure 1) to their outputs, which ensures that they have a uniform distribution.

Note that if $\ell \in O(\log \kappa)$ then the hash function H_{AB} has a polynomial-sized domain, so can be described as a lookup table provided as a common input to the protocol by both parties. At this stage we do not make any assumptions about H_{AB}; this means that the leakage in the protocol will depend on the hash function, so its description is also passed to the functionality $\mathcal{F}_{\text{Leaky-2-Mult}}$ (Figure 3). We require H_{AB} to take as additional input an index $k \in [r]$ and a bit in $\{0,1\}$, to provide independence between different uses, and our later protocols require the function to be different in protocol instances between different pairs of parties (we use the notation H_{AB} to emphasize this).

Functionality $\mathcal{F}^{r,\ell}_{\text{Leaky-2-Mult}}$

INPUT: $(x_1, \ldots, x_r) \in \mathbb{F}_2^r$ from P_A and $(y_1, \ldots, y_r) \in \mathbb{F}_2^r$ from P_B.
COMMON INPUT: A hash function $\mathsf{H}_{AB} : [r] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}$.

1. Sample $\boldsymbol{z}^A, \boldsymbol{z}^B \leftarrow \mathbb{F}_2^r$ such that $\boldsymbol{z}^A + \boldsymbol{z}^B = \boldsymbol{x} * \boldsymbol{y}$ (where $*$ denotes component-wise product).
2. Output $\boldsymbol{z}^A$ to P_A and $\boldsymbol{z}^B$ to P_B.

Leakage: If P_B is corrupt:

1. Let $\mathbf{H} \in \mathbb{F}_2^{r \times 2^\ell}$ be defined so that entry (k, k') of $\mathbf{H}$ is $\mathsf{H}_{AB}(k, 1 \oplus y_k, \boldsymbol{t}_k \oplus k')$, where $\boldsymbol{t}_k \leftarrow \{0,1\}^\ell$.
2. Sample a random unit vector $\boldsymbol{e} \in \mathbb{F}_2^{2^\ell}$ and send $(\mathbf{H}, \boldsymbol{u} = \mathbf{H}\boldsymbol{e} + \boldsymbol{x})$ to $\mathcal{A}$.

Fig. 3. Ideal functionality for leaky secret-shared two-party bit multiplication

Leakage. We now analyse the exact security of the protocol in Figure 4 when using short keys, and explain how this is specified in the functionality $\mathcal{F}_{\text{Leaky-2-Mult}}$ (Figure 3). Since a random share of zero is added to the outputs, note that the output distribution is uniformly random. Also, like IKNP, the protocol is *perfectly*

secure against a corrupt P_A (or sender), so we only need to be concerned with leakage to a corrupt P_B who also sees the intermediate values of the protocol.

The leakage is different for each k, depending on whether $y_k = 0$ or $y_k = 1$, so we consider the two cases separately. Within each case, there are two potential sources of leakage: firstly, the corrupt P_B's knowledge of $\boldsymbol{t}_k$ and ρ_k may cause leakage (where ρ_k is a random share of zero), since these values are used to define P_A's output. Secondly, the d_k values seen by P_B, which equal

$$d_k = \mathsf{H}_{AB}(k, y_k, \boldsymbol{t}_k) \oplus \mathsf{H}_{AB}(k, 1 \oplus y_k, \boldsymbol{t}_k \oplus \Delta) \oplus x_k, \tag{1}$$

may leak information on P_A's inputs x_k.

Case 1 ($y_k = 1$). In this case there is only leakage from the values $\boldsymbol{t}_k$ and ρ_k, which are used to define P_A's output. Since $z_k^A = \mathsf{H}_{AB}(k, 0, \boldsymbol{t}_k \oplus \Delta) \oplus \rho_k$, all of P_A's outputs (and hence, also inputs) where $y_k = 1$ effectively have only ℓ bits of min-entropy in the view of P_B, corresponding to the random choice of Δ. In this case P_B's output is $z_k^B = z_k^A \oplus x_k = \mathsf{H}_{AB}(k, 0, \boldsymbol{t}_k \oplus \Delta) \oplus \rho_k \oplus x_k$. To ensure that P_B's view is simulable the functionality needs to sample a random string $\Delta \leftarrow \{0,1\}^\ell$ and leak $\mathsf{H}_{AB}(k, 0, \boldsymbol{t}_k \oplus \Delta) \oplus x_k$ to a corrupt P_B.

Concerning the d_k values, notice that when $y_k = 1$ P_B can compute $\mathsf{H}_{AB}(k, 1, \boldsymbol{t}_k)$ and use (1) to recover $\mathsf{H}_{AB}(k, 0, \boldsymbol{q}_k) + x_k$, which equals $z_k^A + \rho_k + x_k$. However, this is not a problem, because in this case we have $z_k^B = z_k^A + x_k$, so d_k can be simulated given P_B's output.

Case 2 ($y_k = 0$). Here the d_k values seen by P_B causes leakage on P_A's inputs, because Δ is short. Looking at (1), d_k leaks information on x_k because $\Delta \leftarrow \{0,1\}^\ell$ is the only unknown in the equation, and is fixed for every k. Similarly to the previous case, this means that all of P_A's inputs where $y_k = 0$ have only ℓ bits of min-entropy in the view of an adversary who corrupts P_B. We can again handle this leakage, by defining $\mathcal{F}_{\text{Leaky-2-Mult}}$ to leak $\mathsf{H}_{AB}(k, 1, \boldsymbol{t}_k \oplus \Delta) + x_k$ to a corrupt P_B.

Note that there is no leakage from the $\boldsymbol{t}_k$ values when $y_k = 0$, because then $\boldsymbol{t}_k = \boldsymbol{q}_k$, so these messages are independent of Δ and the inputs of P_A.

In the functionality $\mathcal{F}_{\text{Leaky-2-Mult}}$, we actually modify the above slightly so that the leakage is defined in terms of linear algebra, instead of the hash function H_{AB}, to simplify the translation to the DRSD problem later on. Therefore, $\mathcal{F}_{\text{Leaky-2-Mult}}$ defines a matrix $\mathbf{H} \in \mathbb{F}_2^{r \times 2^\ell}$, which contains the 2^ℓ values $\{\mathsf{H}_{AB}(k, 1 \oplus y_k, \boldsymbol{t}_k \oplus \Delta)\}_{\Delta \in \{0,1\}^\ell}$ in row k, where each $\boldsymbol{t}_k$ is uniformly random. Given $\mathbf{H}$, the leakage from the protocol can then be described by sampling a random unit vector $\boldsymbol{e} \in \mathbb{F}_2^{2^\ell}$ (which corresponds to $\Delta \in \{0,1\}^\ell$ in the protocol) and leaking $\boldsymbol{u} = \mathbf{H}\boldsymbol{e} + \boldsymbol{x}$ to a corrupt P_B.

Communication complexity. The cost of computing r secret-shared products is that of ℓ random, correlated OTs on r-bit strings, and a further r bits of communication. Using OT extension [IKNP03, ALSZ13] to implement the correlated OTs the amortized cost is $\ell(r + \kappa)$ bits, for computational security κ. This gives a total cost of $\ell(r + \kappa) + r$ bits.

Theorem 3.1 *Protocol $\Pi^{r,\ell}_{\text{Leaky-2-Mult}}$ securely implements the functionality $\mathcal{F}^{r,\ell}_{\text{Leaky-2-Mult}}$ with perfect security in the $(\mathcal{F}_{\Delta\text{-ROT}}, \mathcal{F}_{\text{Zero}})$-hybrid model in the presence of static honest-but-curious adversaries.*

Proof. The main challenge in the proof consists of showing that the leakage to P_B in the functionality can be translated directly to the leakage introduced in the protocol in the view of P_B. More formally, for the two

Protocol $\Pi^{r,\ell}_{\text{Leaky-2-Mult}}$

PARAMETERS: r, number of multiplications; ℓ, key length.
INPUT: $\boldsymbol{x} = (x_1, \ldots, x_r) \in \mathbb{F}_2^r$ from P_A and $\boldsymbol{y} = (y_1, \ldots, y_r) \in \mathbb{F}_2^r$ from P_B.
COMMON INPUT: A hash function $\mathsf{H}_{AB} : [r] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}$.

1. P_A and P_B invoke $\mathcal{F}^{r,\ell}_{\Delta\text{-ROT}}$ where P_A is sender with a random input $\Delta \leftarrow \{0,1\}^\ell$, and P_B is receiver with inputs $(y_1, \ldots, y_r)$. P_A receives random strings $\boldsymbol{q}_k \in \{0,1\}^\ell$ and P_B receives $\boldsymbol{t}_k = \boldsymbol{q}_k \oplus y_k \cdot \Delta$, for $k \in [r]$.
2. Call $\mathcal{F}^r_{\text{Zero}}$ so that P_A and P_B obtain the same random $\rho_k \in \{0,1\}$ for every $k \in [r]$.
3. For each $k \in [r]$, P_A privately sends to P_B:
$$d_k = \mathsf{H}_{AB}(k, 0, \boldsymbol{q}_k) + \mathsf{H}_{AB}(k, 1, \boldsymbol{q}_k + \Delta) + x_k.$$
4. P_B outputs
$$z_k^B = \mathsf{H}_{AB}(k, y_k, \boldsymbol{t}_k) + y_k \cdot d_k + \rho_k, \quad \text{for } k \in [r].$$
5. P_A outputs
$$z_k^A = \mathsf{H}_{AB}(k, 0, \boldsymbol{q}_k) + \rho_k, \quad \text{for } k \in [r].$$

Fig. 4. Leaky secret-shared two-party bit multiplication protocol

cases of a corrupt P_A, and a corrupt P_B, we define a simulator who obtains the corrupted party's inputs and the output of $\mathcal{F}_{\text{Leaky-2-Mult}}$, and simulates the view of the corrupted party during a protocol execution.

No corruptions. Here, no simulation is necessary because all communication is over private channels, so we just need to show that the outputs of an honest execution are distributed identically to the functionality. By inspection, the protocol is correct. Observe that the outputs of P_A are uniformly random, because ρ_k is uniformly random. Since P_B's outputs are fixed by the inputs and P_A's outputs, we are done.

Corrupt P_A. This is the simpler of the remaining two cases. The simulator $\mathcal{S}_A$ receives P_A's inputs $x_1, \ldots, x_r \in \mathbb{F}_2$, as well as the outputs $z_1^A, \ldots, z_r^A$ from $\mathcal{F}_{\text{Leaky-2-Mult}}$. It completes the view of P_A by sampling the $q_1, \ldots, q_r \leftarrow \{0,1\}^\ell$ P_A receives from $\mathcal{F}_{\Delta\text{-ROT}}$, and then sends $\rho_k = z_k^A - \mathsf{H}_{AB}(k, 0, q_k)$ to simulate P_A's outputs from $\mathcal{F}_{\text{Zero}}$.

It is easy to see that the views in the two executions are identically distributed, since no messages are sent to P_A during the protocol, and the definition of ρ_k in the simulation ensures that ρ_k is uniformly random (because z_k^A is) and also consistent with P_A's output and the hash function, as in the protocol.

Corrupt P_B. We define a simulator $\mathcal{S}_B$, who receives the inputs $y_1, \ldots, y_r \in \{0,1\}$, and then obtains the values $z_1^B, \ldots, z_r^B, \mathbf{H}, \boldsymbol{u} = (u_1, \ldots, u_r)$ from the functionality.

Let $\mathcal{S}_B$ sample values $t_1, \ldots, t_r \in \{0,1\}^\ell$ at random, subject to the constraint that for every $k \in [r]$ and $k' \in \{0,1\}^\ell$, $\mathsf{H}_{AB}(k, 1 \oplus y_k, t_k \oplus k')$ is equal to entry (k, k') of $\mathbf{H}$ (viewing k' also as an integer in $[2^\ell]$). Note that because of the way $\mathbf{H}$ is defined in $\mathcal{F}_{\text{Leaky-2-Mult}}$, such a t_k is guaranteed to exist and can be found by searching all $2^{2\ell} = \mathsf{poly}(\kappa)$ possibilities of k' and t_k. This also ensures it will be identically distributed to the t_k sampled by the functionality. $\mathcal{S}_B$ sends these values t_k as the outputs of $\mathcal{F}_{\Delta\text{-ROT}}$ to P_B.

For all $k \in [r]$, $\mathcal{S}_B$ then emulates the output of $\mathcal{F}_{\text{Zero}}$ to P_B as follows:

1. If $y_k = 0$, send $\rho_k = z_k^B + \mathsf{H}_{AB}(k, 0, t_k)$.
2. If $y_k = 1$, send $\rho_k = z_k^B + u_k$.

Finally, for $k \in [r]$, $\mathcal{S}_B$ sends $d_k = u_k + \mathsf{H}_{AB}(k, y_k, t_k)$ to P_B. This completes the simulation of P_B's view.

Regarding indistinguishability, first note that, as observed above, the t_k values are identically distributed in both executions. Now considering the case when $y_k = 0$, we have:

$$z_k^B + \mathsf{H}_{AB}(k, 0, t_k) + \rho_k = 0,$$

from the definition of ρ_k. Since in both worlds z_k^B, t_k and ρ_k are all uniformly random, subject to the above, this means that these values are identically distributed in both worlds. Also, it is easy to see that the simulated d_k values are computed exactly as in the protocol, because of the way $\mathcal{F}_{\text{Leaky-2-Mult}}$ computes u_k.

When $y_k = 1$, we have:

$$\begin{aligned} z_k^B + \mathsf{H}_{AB}(k, 1, t_k) + d_k + \rho_k = 0 &\iff (\rho_k + u_k) + \mathsf{H}_{AB}(k, 1, t_k) + \rho_k = d_k \\ &\iff \mathsf{H}_{AB}(k, 0, t_k \oplus \tilde{\Delta}) + \mathsf{H}_{AB}(k, 1, t_k) + x_k = d_k, \end{aligned}$$

where $\tilde{\Delta} \in [2^\ell]$ denotes the position of the 1 in $\boldsymbol{e}$ sampled by $\mathcal{F}_{\text{Leaky-2-Mult}}$ to compute $\boldsymbol{u}$, so is identically distributed to $\Delta \in \{0,1\}^\ell$ in the real protocol. Therefore, the last equation above holds, which implies that z_k^B, ρ_k and d_k are all distributed identically to the values in the real protocol. □

3.2 MPC for Binary Circuits From Leaky OT

We now show how to use the leaky OT protocol to compute multiplication triples over $\mathbb{F}_2$, using a GMW-style protocol [GMW87, Gol04] optimized for the case of at least h honest parties. This can then be used to obtain a general MPC protocol for binary circuits using Beaver's method [Bea92].

Triple generation. We implement the triple generation functionality over $\mathbb{F}_2$, shown in Figure 5. Recall that to create a triple using the GMW method, first each party locally samples shares $x^i, y^i \leftarrow \mathbb{F}_2$. Next, the parties compute shares of the product based on the fact that:

$$(\sum_{i=1}^{n} x^i) \cdot (\sum_{i=1}^{n} y^i) = \sum_{i=1}^{n} x^i y^i + \sum_{i=1}^{n} \sum_{j \neq i} x^i y^j.$$

where x^i denotes P_i's share of $x = \sum_i x^i$.

Since each party can compute $x^i y^i$ on its own, in order to obtain additive shares of $z = xy$ it suffices for the parties to obtain additive shares of $x^i y^j$ for every pair $i \neq j$. This is done using oblivious transfer between P_i and P_j, since a 1-out-of-2 OT implies two-party secret-shared bit multiplication. Due to efficiency considerations we realize a slight variation of this functionality where two (possibly overlapping) subsets $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$ such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party, choose the respective shares of x and y.

Functionality $\mathcal{F}^r_{\text{Triple}}$

1. Sample $(x_j^i, y_j^i, z_j^i) \leftarrow \mathbb{F}_2^3$, for $i \in [n]$ and $j \in [r]$, subject to the constraint that

$$\sum_i z_j^i = (\sum_i x_j^i) \cdot (\sum_i y_j^i)$$

2. Output (x_j^i, y_j^i, z_j^i) to party P_i, for $j \in [r]$.

Fig. 5. Multiplication triple generation functionality.

If we use the *leaky* two-party batch multiplication protocol from the previous section, this approach fails to give a secure protocol because the leakage in $\mathcal{F}_{\text{Leaky-2-Mult}}$ allows a corrupt P_B to guess P_A's inputs with probability $2^{-\ell}$. When using this naively, P_A carries out a secret-shared multiplication using the *same input shares* with every other party, which allows every corrupt party to attempt to guess P_A's shares, increasing the success probability further. If the number of corrupted parties is not too small then this gives the adversary a significant chance of successfully guessing the shares of *every honest party*, completely breaking security.

To avoid this issue, we require P_A to *randomize* the shares used as input to $\mathcal{F}_{\text{Leaky-2-Mult}}$, in such a way that we still preserve correctness of the protocol. To do this, the parties will use $\mathcal{F}_{\text{Zero}}$ to generate random zero shares $s^{i,j} \in \mathbb{F}_2$ (held by P_i), satisfying $\sum_i s^{i,j} = 0$ for all $j \in [n]$, and then P_i and P_j will multiply $x^i + s^{i,j}$ and y^j. This means that all parties end up computing shares of:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(x^i + s^{i,j})y^j = \sum_{j=1}^{n} y^j \sum_{i=1}^{n}(x^i + s^{i,j}) = xy,$$

so still obtain a correct triple.

Finally, to ensure that the output shares are uniformly random, fresh shares of zero will be added to each party's share of xy. Note that masking each x^i input to $\mathcal{F}_{\text{Leaky-2-Mult}}$ means that it doesn't matter if the individual shares are leaked to the adversary, as long as it is still hard to guess the *sum of all shares*. This means that we only need to be concerned with the *sum of the leakage* from $\mathcal{F}_{\text{Leaky-2-Mult}}$. Recall that each individual instance leaks the input of an honest party P_i masked by $\mathbf{H}_i\boldsymbol{e}_i$, where $\mathbf{H}_i$ is a random matrix and $\boldsymbol{e}_i \in \mathbb{F}_2^{2^\ell}$ is a random unit vector. Summing up all the leakage from h honest parties, we get

$$\sum_{i=1}^{h} \mathbf{H}_i\boldsymbol{e}_i = (\mathbf{H}_1\| \cdots \|\mathbf{H}_h) \begin{pmatrix} \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{e}_h \end{pmatrix}$$

This is exactly an instance of the $\mathsf{DRSD}_{r,h,\ell}$ problem, so is pseudorandom for an appropriate choice of parameters.

We remark that the number of triples generated, r, affects the hardness of DRSD. However, we can create an arbitrary number of triples without changing the assumption by repeating the protocol for a fixed r.

Reducing the number of OT channels. The above approach reduces communication of GMW by a factor κ/ℓ, for ℓ-bit keys, but still requires a complete network of $n(n-1)$ OT and communication channels between the parties. We can reduce this further by again taking advantage of the fact that there are at least h honest parties. We observe that when using our two-party secret-shared multiplication protocol to generate triples, information is only leaked on the x^i shares, and not the y^i shares of each triple. This means that $h-1$ parties can choose their shares of y to be zero, and y will still be uniformly random to an adversary who corrupts up to $t = n - h$ parties. This reduces the number of OT channels needed from $n(n-1)$ to $(t+1)(n-1)$.

When the number of parties is large enough, we can do even better using *random committees*. We randomly choose two committees, $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$, such that except with negligible probability, $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party. Only the parties in $\mathcal{P}_{(h)}$ choose non-zero shares of x, and parties in $\mathcal{P}_{(1)}$ choose non-zero shares of y; all other parties do not take part in any OT instances, and just output random sharings of zero. We remark that it can be useful to choose the parameter h *lower than* the actual number of honest parties, to enable a smaller committee size (at the cost of potentially larger

keys). When the total number of parties, n, is large enough, this means the number of interacting parties can be independent of n. The complete protocol, described for two fixed committees satisfying our requirements, is shown in Figure 6.

Protocol Π^r_{Triple}

The protocol runs between a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, containing two (possibly overlapping) subsets $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$, such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party. We denote $n_h = |\mathcal{P}_{(h)}|$, $n_1 = |\mathcal{P}_{(1)}|$.

CRS: Random hash functions $\mathsf{H}_i : [r] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}$, for $i \in [n_h]$.

1. Each party $P_i \in \mathcal{P}_{(h)}$ samples $x^i_k \leftarrow \mathbb{F}_2$, and each $P_j \in \mathcal{P}_{(1)}$ samples $y^j_k \leftarrow \mathbb{F}_2$, for $k \in [r]$.
2. Call $\mathcal{F}_{\text{Zero}}$ so that each $P_\tau \in \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ obtains shares $(\rho^\tau_1, \ldots, \rho^\tau_r)$ and each $P_i \in \mathcal{P}_{(h)}$ obtains shares $(s^{i,j}_1, \ldots, s^{i,j}_r)_{j \in \mathcal{P}_{(1)}}$, such that $\bigoplus_{\tau \in \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}} \rho^\tau_k = 0$ and $\bigoplus_{i \in \mathcal{P}_{(h)}} s^{i,j}_k = 0$.
3. Every pair $(P_i, P_j) \in \mathcal{P}_{(h)} \times \mathcal{P}_{(1)}$ runs $\mathcal{F}^{r,\ell}_{\text{Leaky-2-Mult}}(\mathsf{H}_i)$ on input $\{x^i_k + s^{i,j}_k\}_{k \in [r]}$ from P_i and $\{y^j_k\}_{k \in [r]}$ from P_j. For $k \in [r]$, P_i receives $a^{i,j}_k$ and P_j receives $b^{j,i}_k$ such that $a^{i,j}_k + b^{j,i}_k = (x^i_k + s^{i,j}_k) \cdot y^j_k$.
4. Each $P_i \in \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ computes, for $k \in [r]$:

$$z^i_k = (x^i_k + s^{i,i}_k) \cdot y^i_k + \sum_{j \neq i} (a^{i,j}_k + b^{i,j}_k) + \rho^i_k$$

   where if any value $x^i_k, y^i_k, a^{i,j}_k, b^{i,j}_k, s^{i,i}_k$ has not been defined by P_i, it is set to zero.
5. P_i outputs the shares $(x^i_k, y^i_k, z^i_k)_{k \in [r]}$.

Fig. 6. Secret-shared triple generation using leaky two-party multiplication.

Communication complexity. Recall from the analysis in Section 3.1 that when using protocol $\Pi_{\text{Leaky-2-Mult}}$ with Π_{Triple}, the cost of computing r secret-shared triples is that of ℓ random, correlated OTs on r-bit strings, and a further r bits of communication between every pair of parties. This gives a total cost of $\ell(r+\kappa)+r$ bits between every pair of parties who has an OT channel (ignoring $\mathcal{F}_{\text{Zero}}$ and the seed OTs for OT extension, since their communication cost is independent of the number of triples). If the two committees $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$ have sizes $n_h \leq n$ and $n_1 \leq t+1$ then we have the following theorem

Theorem 3.2 *Protocol Π_{Triple} securely realizes $\mathcal{F}^r_{\text{Triple}}$ in the $(\mathcal{F}^{r,\ell}_{\text{Leaky-2-Mult}}, \mathcal{F}^{(n+1)r}_{\text{Zero}})$-hybrid model, based on the $\mathsf{DRSD}_{r,h,\ell}$ assumption, where h is the number of honest parties in $\mathcal{P}_{(h)}$. The amortized communication cost is $\leq n_h n_1(\ell + \ell\kappa/r + 1)$ bits per triple.*

Proof. The claimed communication complexity follows from the previous analysis. Security relies on the fact that $P_i \in \mathcal{P}_{(h)}$'s input to $\mathcal{F}_{\text{Leaky-2-Mult}}$ is always of the form $x^i + s^{i,j}$, where $s^{i,j}$ is a fresh, random sharing of zero. This means that any leakage on P_i's input from $\mathcal{F}_{\text{Leaky-2-Mult}}$ is perfectly masked by $s^{i,j}$, and we only need to consider the *sum* of the leakage from all honest parties in $\mathcal{P}_{(h)}$.

Recall that we have two committees $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$ of sizes n_h and n_1, with at least h and 1 honest parties, respectively. Let $\mathcal{A}$ be an adversary corrupting a set of parties A. Throughout the proof we will write $x_1, \ldots, x_r$ to denote the components of a vector $\boldsymbol{x} \in \mathbb{F}_2^r$.

We construct a simulator, $\mathcal{S}$, which interacts with $\mathcal{A}$ as follows:

1. Simulate the CRS with n_h randomly sampled functions $\mathsf{H}_i : [r] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}$.
2. Call $\mathcal{F}_{\text{Triple}}$ to receive the corrupted parties' outputs, $(x^i_k, y^i_k, z^i_k)_{i \in A \cap (\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}), k \in [r]}$.

3. For each $i \in \mathcal{P}_{(h)} \cap A$, sample $\boldsymbol{s}^{i,j} \leftarrow \mathbb{F}_2^r$, for $j \in [n_1]$, and send these to $\mathcal{A}$ as the shares output by $\mathcal{F}_{\text{Zero}}$.
4. Let $P_i \in \mathcal{P}_{(h)}, P_j \in \mathcal{P}_{(1)}$. Compute the messages that would be sent by $\mathcal{F}_{\text{Leaky-2-Mult}}$ to the adversary as follows:
   (a) $P_i, P_j \in A$: Using both parties' inputs, generate their random output shares as $\mathcal{F}_{\text{Leaky-2-Mult}}$ would do and send these to $\mathcal{A}$. Explicitly, the simulator samples shares $\boldsymbol{a}^{i,j}, \boldsymbol{b}^{j,i}$ that sum to $(\boldsymbol{x}^i + \boldsymbol{s}^{i,j}) * \boldsymbol{y}^j$, and the leakage $(\mathbf{H}^{i,j}, \boldsymbol{u}^{i,j})$ on $\boldsymbol{x}^i + \boldsymbol{s}^{i,j}$ (just as $\mathcal{F}_{\text{Leaky-2-Mult}}$ would do).
   (b) $P_i \in A, P_j \notin A$: Emulate the corrupt P_i's view honestly, by sampling $\boldsymbol{a}^{i,j} \leftarrow \mathbb{F}_2^r$.
   (c) $P_i \notin A, P_j \in A$: Sample uniform values $\boldsymbol{b}^{j,i}, \boldsymbol{u}^{i,j} \leftarrow \mathbb{F}_2^r$, and sample $\mathbf{H}^{i,j} \in \mathbb{F}_2^{m \times 2^\ell}$ exactly as $\mathcal{F}_{\text{Leaky-2-Mult}}$ would do, using knowledge of H_i and $\boldsymbol{y}^j$.
5. For $i \in A \cap (\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)})$, compute $\boldsymbol{\rho}^i = \boldsymbol{z}^i + (\boldsymbol{x}^i + \boldsymbol{s}^{i,i}) * \boldsymbol{y}^i + \sum_{j \neq i} (\boldsymbol{a}^{i,j} + \boldsymbol{b}^{i,j})$ and send this as the ρ_k^i share from $\mathcal{F}_{\text{Zero}}$.
6. Send to $\mathcal{A}$ the values $\{\boldsymbol{a}^{i,j}\}_{i \in \mathcal{P}_{(h)} \cap A, j \in \mathcal{P}_{(1)}} \cup \{\boldsymbol{b}^{j,i}, \mathbf{H}^{i,j}, \boldsymbol{u}^{i,j}\}_{i \in \mathcal{P}_{(h)}, j \in \mathcal{P}_{(1)} \cap A}$ as defined above, to simulate the outputs of $\mathcal{F}_{\text{Leaky-2-Mult}}$.

We first consider the distribution of the parties' outputs.

Claim 3.1 *The outputs of the protocol are distributed identically to outputs of the functionality.*

Proof. We need to to show that, in the real protocol, $\{z_k^i\}_{i,k}$ are uniformly random subject to $\sum_i z_k^i = \sum_i x_k^i \cdot \sum_i y_k^i$. Firstly, the correctness constraint holds because

$$\begin{aligned}
\sum_{i=1}^n z_k^i &= \sum_{i=1}^n \left((x_k^i + s_k^{i,i}) \cdot y_k^i + \sum_{j \neq i} (a_k^{i,j} + b_k^{i,j}) + \rho_k^i \right) \\
&= \sum_{i=1}^n x_k^i \cdot y_k^i + \sum_{i=1}^n \left(y_k^i \cdot s_k^{i,i} + \sum_{j \neq i} (a_k^{i,j} + b_k^{j,i}) \right) \\
&= \sum_{i=1}^n x_k^i \cdot y_k^i + \sum_{i=1}^n \left(y_k^i \cdot s_k^{i,i} + \sum_{j \neq i} y_k^i \cdot (x_k^j + s_k^{j,i}) \right) \\
&= x_k \cdot y_k + \sum_{i=1}^n y_k^i \cdot \sum_{j=1}^n s_k^{j,i} \\
&= x_k \cdot y_k
\end{aligned}$$

where the second line above holds because $\sum_i \rho_k^i = 0$, and the final line uses $\sum_j s_k^{j,i} = 0$.

Now, to see that $(z_k^i)_i$ are *uniformly random*, subject to the above, notice that the masks $(\rho_k^i)_{i=1}^{n-1}$ are uniformly random in the protocol, so the same is true of $(z_k^i)_{i=1}^{n-1}$. This completes the claim. □

We next consider the entire view of the environment $\mathcal{Z}$, which is the joint distribution of all parties' inputs and outputs, and the messages received by the adversary during the protocol. Using vector notation, this is:

$$(\boldsymbol{x}^i, \boldsymbol{y}^j, \boldsymbol{z}^i, \boldsymbol{z}^j)_{i \in \mathcal{P}_{(h)}, j \in \mathcal{P}_{(1)}}, (\boldsymbol{\rho}^i, \boldsymbol{s}^{i,j}, \boldsymbol{a}^{i,j})_{i \in A \cap \mathcal{P}_{(h)}, j \in \mathcal{P}_{(1)}}, (\boldsymbol{\rho}^j, \boldsymbol{b}^{j,i}, \boldsymbol{u}^{i,j}, \mathbf{H}^{i,j})_{i \in \mathcal{P}_{(h)}, j \in \mathcal{P}_{(1)} \cap A}$$

First note that the $\boldsymbol{\rho}^\tau, \tau \in \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ and $\boldsymbol{s}^{i,j}$ shares, for $i \in \mathcal{P}_{(h)} \cap A, j \in \mathcal{P}_{(1)} \setminus A$, are uniformly random in both executions, since the environment never sees the honest parties' shares. Secondly, recall that in the simulation, $\boldsymbol{a}^{i,j}$ for corrupt P_i and honest P_j) and $\boldsymbol{b}^{j,i}$ (for corrupt P_j and honest P_i) are computed uniformly at random, and this is identically distributed to the values in the protocol sampled by $\mathcal{F}_{\text{Leaky-2-Mult}}$, because the outputs of the honest party in that instance are not seen by $\mathcal{Z}$. Also, notice that when P_j is corrupt $\mathcal{S}$ computes $\mathbf{H}^{i,j}$ exactly as in the real protocol, because $\mathcal{S}$ knows P_j's input $\boldsymbol{y}^j$.

This leaves the $\{\boldsymbol{u}^{i,j}\}_{i \in \mathcal{P}_{(h)} \setminus A, j \in \mathcal{P}_{(1)} \cap A}$ values, which are the main challenge, because the simulation computes these with random values, whilst the real execution uses the honest P_i's inputs, computing $\boldsymbol{u}^{i,j} = \mathbf{H}^{i,j}\boldsymbol{e}^{i,j} + \boldsymbol{x}^i + \boldsymbol{s}^{i,j}$ for a random unit vector $\boldsymbol{e}^{i,j}$. Let $P_{i_1}, \ldots, P_{i_h}$ be the honest parties in $\mathcal{P}_{(h)}$. Because the $\boldsymbol{s}^{i,j}$ values are random shares of zero, it holds that the partial views containing the entire transcript *except for* $(\boldsymbol{u}^{i_1,j})_{j \in \mathcal{P}_{(1)} \cap A}$ are identically distributed. This is because for $P_j \in \mathcal{P}_{(1)} \cap A$, the masks $\boldsymbol{s}^{i_2,j}, \ldots, \boldsymbol{s}^{i_h,j}$ in the protocol are random and independent of the view of $\mathcal{Z}$, which makes the corresponding $\boldsymbol{u}^{i_2,j}, \ldots, \boldsymbol{s}^{i_h,j}$ values distributed the same as in the simulation.

Once we include $\boldsymbol{u}^{i_1,j}$, however, these values are no longer independent because $\sum_{i \in \mathcal{P}_{(h)}}^{n} \boldsymbol{s}^{i,j} = 0$. We therefore look at the distribution of $\sum_{k=1}^{h} \boldsymbol{u}^{i_k,j}$, for some fixed $j \in \mathcal{P}_{(1)} \cap A$. In the protocol, we have

$$\sum_{i \in \mathcal{P}_{(h)} \setminus A} \boldsymbol{u}^{i,j} = \sum_{i \in \mathcal{P}_{(h)} \setminus A} (\boldsymbol{x}^i + \boldsymbol{s}^{i,j} + \mathbf{H}^{i,j}\boldsymbol{e}^{i,j})$$

for some random, weight-1 vector $\boldsymbol{e}^{i,j}$. In the simulation, all of the $\boldsymbol{u}^{i,j}$'s are uniformly random.

Since $\mathcal{Z}$ can compute $\sum_{i \in \mathcal{P}_{(h)} \setminus A}(\boldsymbol{x}^i + \boldsymbol{s}^{i,j})$ with the information it already has, it follows that distinguishing the two executions requires distinguishing

$$\mathbf{H}_i\boldsymbol{e}_i := \left(\mathbf{H}^{i_1,j} \| \mathbf{H}^{i_2,j} \| \cdots \| \mathbf{H}^{i_h,j}\right) \begin{pmatrix} \boldsymbol{e}^{i_1,j} \\ \boldsymbol{e}^{i_2,j} \\ \vdots \\ \boldsymbol{e}^{i_h,j} \end{pmatrix}$$

and the uniform distribution on r bits (given $\mathbf{H}_i$).

We claim that this corresponds exactly to solving the $\mathsf{DRSD}_{r,h,\ell}$ problem, because $\mathbf{H}_i$ is uniformly distributed in $\mathbb{F}_2^{r \times 2^\ell h}$ and $\boldsymbol{e}_i$ is a uniformly random, 1-regular error vector of weight h.

Lemma 3.2 *Any environment distinguishing the real and ideal executions with advantage δ can be used to break $\mathsf{DRSD}_{r,h,\ell}$ with advantage at least δ/t (where $t = |\mathcal{P}_{(1)} \cap A|$).*

Proof. Assume w.l.o.g. that the corrupted parties in $\mathcal{P}_{(1)}$ are indexed $P_1, \ldots, P_t$. We construct a sequence of hybrid executions, $\mathbf{HYB}_0, \ldots, \mathbf{HYB}_t$, where hybrid $\mathbf{HYB}_0$ is identical to the simulation. In hybrid $\mathbf{HYB}_{j'}$, instead of the simulator sampling $\boldsymbol{u}^{i,j}$ (for $j \leq j', i \in \mathcal{P}_{(h)} \setminus A$) at random, we replace this with the real $\boldsymbol{u}^{i,j}$ generated using P_i's inputs as in the protocol. The final hybrid $\mathbf{HYB}_t$ is therefore identically distributed to the real execution.

Let $\mathcal{A}$ be an adversary for which the environment $\mathcal{Z}$ distinguishes between $\mathbf{HYB}_{j'}$ and $\mathbf{HYB}_{j'+1}$ with advantage δ, for some index $j' < t$. We construct a distinguisher $\mathcal{D}$ for $\mathsf{DRSD}_{r,h,\ell}$ as follows. $\mathcal{D}$ receives a DRSD challenge $\mathbf{H}_{j'} \in \mathbb{F}_2^{r \times h2^\ell}, \boldsymbol{c}^{j'} \in \mathbb{F}_2^r$. Write $\mathbf{H}_{j'} = (\mathbf{H}^{i_1,j'} \| \mathbf{H}^{i_2,j'} \| \cdots \| \mathbf{H}^{i_h,j'})$, where each $\mathbf{H}^{i_k,j'} \in \mathbb{F}_2^{r \times 2^\ell}$. Now $\mathcal{D}$ simulates an execution of Π_{Triple} with $\mathcal{A}$ as $\mathcal{S}$ would, with the following differences.

- $\mathcal{D}$ samples a set of honest parties' shares, $(\boldsymbol{x}^i, \boldsymbol{y}^i, \boldsymbol{z}^i)_{i \notin A}$ which, together with the corrupt parties' shares known to $\mathcal{D}$, form correct triples.

– Instead of sampling the function H_i in the CRS at random, sample it such that for every $k \in [h]$, the matrix $\mathbf{H}^{i_k,j'}$, sent later to the corrupt $P_{j'}$, is equal to the challenge matrix $\mathbf{H}^{i_k,j'}$. (The remainder of the CRS is sampled at random.)
– Instead of sampling the leakage terms $\boldsymbol{u}^{i_k,j'}$ (for $k \in [h]$) uniformly and independently, sample them at random such that $\sum_{i \in \mathcal{P}_{(h)}} (\boldsymbol{u}^{i,j'} + \boldsymbol{x}^i) = \boldsymbol{c}^{j'}$.
– For each $j < j'$, instead of sampling $\boldsymbol{u}^{i_k,j'}$ uniformly, compute them as in the real protocol using the honest parties' shares and shares of zero.

To conclude, $\mathcal{D}$ sends all the output shares to $\mathcal{Z}$ and outputs the same as $\mathcal{Z}$.

If the challenge $(\mathbf{H}_{j'}, \boldsymbol{c}^{j'})$ comes from the DRSD distribution then the $\boldsymbol{u}^{i_k,j'}$ values are distributed as in a real execution, so we are in hybrid $\mathbf{HYB}_{j'+1}$. On the other hand, if $\boldsymbol{c}^{j'}$ is uniformly random then so are the $\boldsymbol{u}^{i_k,j'}$, so we are in $\mathbf{HYB}_{j'}$. Therefore, the advantage of $\mathcal{D}$ is δ, the same as that of $\mathcal{Z}$. A standard hybrid argument then shows that there exists a distinguisher for $\mathbf{HYB}_0$ and $\mathbf{HYB}_t$, which has advantage at least δ/t. □

Parameters for unconditional security. Recall from Lemma 2.1 and Corollary 2.1 that if $\ell = 1$ and $h \geq r + s$, or if $h \geq (r + 2s)/\ell$ for any ℓ, then $\mathsf{DRSD}_{r,h,\ell}$ is *statistically hard*, with statistical security 2^{-s}. This means when h is large enough we can use 1-bit keys, and every pair of parties who communicates only needs to send $2 + \kappa/r$ bits over the network.[6]

MPC using multiplication triples. Our protocol for multiplication triples can be used to construct a semi-honest MPC protocol for binary circuits using Beaver's approach [Bea92]. The parties first secret-share their inputs between all other parties. Then, XOR gates can be evaluated locally on the shares, whilst an AND gate requires consuming a multiplication triple, and two openings with Beaver's method. Each opening can be done with $2(n-1)$ bits of communication as follows: all parties send their shares to P_1, who sums the shares together and sends the result back to every other party.

In the 1-bit key case mentioned above, using two (deterministic) committees of sizes n and $t+1$ and setting, for instance, $r = \kappa$ implies the following corollary. Note that the number of communication channels is $(t+1)(n-1)$ and not $(t+1)n$, because in the deterministic case $\mathcal{P}_{(1)}$ is contained in $\mathcal{P}_{(h)}$, so $t+1$ sets of the shared cross-products can be computed locally.

Corollary 3.3 *Assuming OT and OWF, there is a semi-honest MPC protocol for binary circuits with an amortized communication complexity of no more than* $3(t+1)(n-1)+4(n-1)$ *bits per AND gate, if there are at least* $\kappa + s$ *honest parties.*

4 Multi-Party Garbled Circuits with Short Keys

In this section we present our second contribution: a constant-round MPC protocol based on garbled circuits with short keys. The protocol has two phases, a preprocessing phase independent of the parties' actual inputs where the garbled circuit is mutually generated by all parties, and an online phase where the computation is performed. We first abstractly discuss the details of our garbling method, and then turn to the two protocols for generating and evaluating the garbled circuit.

Functionality $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$

COMMON INPUT: A function $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^{\ell_{\text{BMR}}} \to \{0,1\}^{n\ell_{\text{BMR}}+1}$. Let H' denote the same function excluding the least significant bit of the output.

Let C_f be a boolean circuit with fan-out-one gates. Denote by $\mathsf{AND}, \mathsf{XOR}$ and SPLIT its sets of AND, XOR and Splitter gates, respectively. Given a gate, let I and O be the set of its input and output wires, respectively. If $g \in \mathsf{SPLIT}$, then $I = \{w\}$ and $O = \{u, v\}$, otherwise $O = \{w\}$.
The functionality proceeds as follows $\forall i \in [n]$:

1. $\forall g \in \mathsf{XOR}$, sample $\Delta^i_g \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$.
2. For each circuit-input wire u, sample $\lambda_u \leftarrow \{0,1\}$ and $k^i_{u,0} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$. If u is input to a XOR gate g, set $k^i_{u,1} = k^i_{u,0} \oplus \Delta^i_g$, otherwise $k^i_{u,1} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$.
3. Passing topologically through all the gates $g \in \{\mathsf{AND} \cup \mathsf{XOR} \cup \mathsf{SPLIT}\}$ of the circuit:
   - If $g \in \mathsf{XOR}$:
     - Set $\lambda_w = \bigoplus_{x \in I} \lambda_x$
     - Set $k^i_{w,0} = \bigoplus_{x \in I} k^i_{x,0}$ and $k^i_{w,1} = k^i_{w,0} \oplus \Delta^i_g$
   - If $g \in \mathsf{AND}$:
     - Sample $\lambda_w \leftarrow \{0,1\}$.
     - $k^i_{w,0} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$. If w is input to a XOR gate g' set $k^i_{w,1} = k^i_{w,0} \oplus \Delta^i_{g'}$, else $k^i_{w,1} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$.
     - For $a, b \in \{0,1\}$, representing the public values of wires u and v, let $c = (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w$. Store the four entries of the garbled version of g as:

$$\tilde{\boldsymbol{g}}_{a,b} = \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, b, k^i_{u,a}) \oplus \mathsf{H}(i, a, k^i_{v,b}) \right) \oplus (c, k^1_{w,c}, \ldots, k^n_{w,c}), \quad (a,b) \in \{0,1\}^2.$$

   - If $g \in \mathsf{SPLIT}$:
     - Set $\lambda_x = \lambda_w$ for every $x \in O$.
     - $\forall x \in O$, sample $k^i_{x,0} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$. If $x \in O$ is input to a XOR gate g', set $k^i_{x,1} = k^i_{x,0} \oplus \Delta^i_{g'}$, otherwise $k^i_{x,1} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$.
     - For $c \in \{0,1\}$, the public value on w, store the two entries of the garbled version of g as:

$$\tilde{g}_c = \left(\bigoplus_{i=1}^{n} \mathsf{H}'(i, 0, k^i_{w,c}), \bigoplus_{i=1}^{n} \mathsf{H}'(i, 1, k^i_{w,c}) \right) \oplus (k^1_{u,c}, \ldots, k^n_{u,c}, k^1_{v,c}, \ldots, k^n_{v,c}),\ c \in \{0,1\}$$

4. **Output:** For each circuit-input wire u, send λ_u to the party providing inputs to C_f on u. For every circuit wire v and $i \in [n]$, send $k^i_{v,0}, k^i_{v,1}$ to P_i. Finally, send to all parties $\tilde{g}$ for each $g \in \mathsf{AND} \cup \mathsf{SPLIT}$ and λ_w for each circuit-output wire w.

Fig. 7. Multi-party garbling functionality

4.1 The Multi-Party Garbling Scheme

Our garbling method is defined by the functionality $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$ (Figure 7), which creates a garbled circuit that is given to all the parties. It can be seen as a variant of the multi-party garbling technique by Beaver, Micali and Rogaway [BMR90], known as BMR, which has been used and improved in a recent sequence of works [LPSY15, LSS16, BLO16, HSS17].

The main idea behind BMR is that every party P_i contributes a pair of keys $k_{w,0}^i, k_{w,1}^i \in \{0,1\}^\kappa$ and a share of a wire mask $\lambda_w^i \in \{0,1\}$ for each wire w in the circuit. To garble a gate, the corresponding output wire key from every party is encrypted under the combination of all parties' input wire keys, using a PRF or PRG, so that no single party knows all the keys for a gate. In addition, the free-XOR property can be supported by having each party choose their keys such that $k_{w,0}^i \oplus k_{w,1}^i = \Delta^i$, where Δ^i is a global fixed random string known to P_i.

The main difference between our work and recent related protocols is that we use short keys of length ℓ_{BMR} instead of κ, and then garble gates using a random, expanding function $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^{\ell_{\text{BMR}}} \to \{0,1\}^{n\ell_{\text{BMR}}+1}$. Instead of basing security on a PRF or PRG, we then reduce the security of the protocol to the pseudorandomness of the *sum* of H when applied to each of the honest parties' keys, which is implied by the DRSD problem from Section 2.3. We also use H' to denote H with the least significant output bit dropped, which we use for garbling splitter gates.

To garble an AND gate g with input wires u, v and output wire w, each of the 4 garbled rows $\tilde{g}_{a,b}$, for $(a,b) \in \{0,1\}^2$, is computed as:

$$\tilde{g}_{a,b} = \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, b, k_{u,a}^i) \oplus \mathsf{H}(i, a, k_{v,b}^i) \right) \oplus (c, k_{w,c}^1, \ldots, k_{w,c}^n), \tag{2}$$

where $c = (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w$ and $\lambda_u, \lambda_v, \lambda_w$ are the secret-shared wire masks. Each row can be seen as an encryption of the correct n output wire keys under the corresponding input wire keys of all parties. Note that, for each wire, P_i holds the keys $k_{u,0}^i, k_{u,1}^i$ and an additive share λ_u^i of the wire mask. The extra bit value that H takes as input is added to securely increase the stretch of H when using the same input key twice, preventing a 'mix-and-match' attack on the rows of a garbled gate. The output of H is also extended by an extra bit, to allow encryption of the output wire mask c.[7]

Splitter gates. When relying on the DRSD problem, the reuse of a key in multiple gates degrades parameters and makes the problem easier (as the parameter r grows, the key length must be increased), so we cannot handle circuits with arbitrary fan-out. For this reason, we restrict our exposition of the garbling to fan-out-one circuits with so-called *splitter gates*. This type of gate takes as input a single wire w and provides two output wires u, v, each of them with fresh, independent keys representing the same value carried by the input wire. Converting an arbitrary circuit to use splitter gates incurs a cost of roughly a factor of two in the circuit size (see below).

Splitter gates were previously introduced in [TX03] as a fix for a similar issue in the original BMR paper [BMR90], where the wire "keys" were used as seeds for a PRG in order to garble the gates, so that when a wire was used as input to multiple gates, their garbled versions did not use independent pseudorandom masks. Other recent BMR-style papers avoid this issue by applying the PRF over the gate identifier as well, which produces distinct, independent PRF evaluations for each gate.

[6] Note that we still need computational assumptions for OT and zero sharing in order to implement $\mathcal{F}_{\text{Leaky-2-Mult}}$ and $\mathcal{F}_{\text{Zero}}$.

[7] This only becomes necessary when using short keys — in BMR with full-length keys the parties can recover the wire mask by comparing the output with their own two keys, but this does not work if collisions are possible.

Free-XOR. The Free-XOR [KS08] optimization results in an improvement in both computation and communication for XOR gates where a global fixed random Δ_i is chosen by each party P_i and the input keys are locally XORed, yielding the output key of this gate. We cannot use the standard free-XOR technique [KS08, BLO16] for the same reason discussed above: reusing a single offset across multiple gates would make the DRSD problem easier and not be secure. We therefore introduce a new free-XOR technique (inspired by FleXOR [KMR14]) which, combined with our use of splitter gates, allows garbling XOR gates for free without additional assumptions. For each arbitrary fan-in XOR gate g, each party chooses a different offset Δ_g^i, allowing for a free-XOR computation for wires using keys with that offset. For general circuits, this would normally introduce the problem that the input wires may not have the correct offset, requiring some 'translation' to Δ_g. However, because we restrict to gates with fan-out-one and splitter gates, we know that each input wire to g is not an input wire to any other gate, so we can always ensure the keys use the correct offset without any further changes.

Compiling to fan-out-one circuits with splitter gates. Let C_f be an arbitrary fan-out circuit, with A AND gates and X XOR gates, both with fan-in-two. Let I_{C_f} and O_{C_f} be the number of circuit-input and circuit-output wires, respectively. We will now compute the number S of splitter gates that the compiled circuit needs. First, note that each time a wire w is used as input to another gate or as a circuit-output wire, w's fan-out is increased by one. Each of the AND, XOR gates in the pre-compiled circuit provides a fresh output wire to be used in C_f, while using for its inputs two pre-existing wires in the circuit. Output wires also use one pre-existing wire each, while input wires use no pre-existing wires. This means that, to compile C_f to be a fan-out-one circuit, we need to add up to $(2 \cdot X + 2 \cdot A + O_{C_f}) - (A + X + I_{C_f})$ wires. Each of these missing wires, however, can be created by using a splitter gate in the compiled circuit, since each of these gates uses one wire to generate two fresh new wires. So, putting all the pieces together, the compiled circuit requires $S \leq X + A + O_{C_f} - I_{C_f}$ splitter gates. This gives a close upper bound, as if w is a circuit output wire *and* an input wire of another gate then it is being counted twice rather than once in the formula.

4.2 Protocol and Functionalities for Bit and Bit/String Multiplication

Functionality $\mathcal{F}_{\mathrm{Bit}\times\mathrm{Bit}}$

After receiving $(x^i, y^i) \in \mathbb{F}_2^2$ from each party P_i, sample $z_i \leftarrow \mathbb{F}_2$ such that $\sum_i z^i = (\sum_i x^i) \cdot (\sum_i y^i)$, and send z^i to party P_i.

Fig. 8. Secret-shared bit multiplication functionality

Functionality $\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{BitString}}(P_j)$

After receiving $(x^i, y^i) \in \mathbb{F}_2^2$ from each party P_i, as well as $\Delta \in \mathbb{F}_2^{\ell_{\mathrm{BMR}}}$ from P_j, sample $Z_i \leftarrow \mathbb{F}_2$ such that $\sum_i Z^i = (\sum_i x^i) \cdot \Delta$, and send Z^i to party P_i.

Fig. 9. Secret-shared bit/string multiplication functionality

Even though we could implement both $\mathcal{F}_{\mathrm{Bit}\times\mathrm{Bit}}$ and $\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{BitString}}(P_j)$ using $\mathcal{F}_{\mathrm{Triple}}$, there are more efficient ways to implement the latter: One by building directly from $\mathcal{F}_{\text{Leaky-2-Mult}}$, and another using [ALSZ13].

– $\mathcal{F}_{\text{Leaky-2-Mult}}$-hybrid implementation (Figure 11): As the length-ℓ_{BMR} string R_g^j is not secret-shared and just known to one party, we only need to perform $n-1$ invocations of $\mathcal{F}_{\text{Leaky-2-Mult}}$ in order to multiply it with a secret-shared bit $x = x^1 + \cdots + x^n$. The protocol uses random shares of zero to mask the inputs and outputs of $\mathcal{F}_{\text{Leaky-2-Mult}}$, similarly to the Π_{Triple} protocol.
Note that this does not directly implement the functionality shown in Figure 9, because $\Pi^{r,\ell_{\text{BMR}}}_{\text{Bit}\times\text{String}}$ performs a batch of r independent multiplications in parallel. However, in the protocol $\Pi^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$ all the gates can be garbled in parallel, so a batch version of the functionality (as described in Figure 10) suffices. The amortized communication complexity obtained is $\ell_{\text{BMR}}(1 + \ell_{\text{OT}} + \ell_{\text{OT}}\kappa/r)$ bits.
– [ALSZ13] implementation: The amortized communication complexity is $\kappa + \ell_{\text{BMR}}$ bits.

Functionality $\mathcal{F}^{r,\ell_{\text{BMR}}}_{\text{BitString}}$

After receiving input $(P_j, x_1^i, \ldots, x_m^i)$ from every party P_i, and additional inputs $\Delta_1, \ldots, \Delta_r$ from P_j, where each $x_k^i \in \{0,1\}$ and $\Delta_k \in \{0,1\}^{\ell_{\text{BMR}}}$:

1. Sample $Z_k^i \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$, for $i \in [n]$ and $k \in [r]$, subject to the constraint that
$$\bigoplus_i Z_k^i = \Delta_k \cdot \bigoplus_i x_k^i, \quad \text{for } k \in [r]$$
2. Output $Z_1^i, \ldots, Z_r^i$ to party P_i

Fig. 10. Batch secret-shared bit/string multiplication between P_j and all parties

Protocol $\Pi^{r,\ell_{\text{BMR}}}_{\text{Bit}\times\text{String}}$, n-party Bit/String-Mult

To multiply the strings $\Delta_1, \ldots, \Delta_r \in \{0,1\}^{\ell_{\text{BMR}}}$ held by P_j with secret-shared bits $(x_1^i, \ldots, x_r^i)_{i\in[n]}$:

1. Denote the v-th bit of Δ_k by $\Delta_{k,v}$. For $v \in [\ell_{\text{BMR}}]$:
   (a) Call $\mathcal{F}^{2r}_{\text{Zero}}$ so that each P_i obtains fresh shares $(\rho^i_{1,v}, \ldots, \rho^i_{m,v}, \sigma^i_{1,v}, \ldots, \sigma^i_{m,v})$, such that $\bigoplus_i \rho^i_{k,v} = 0$ and $\bigoplus_i \sigma^i_{k,v} = 0$
   (b) For each $i \neq j$, P_i and P_j run $\mathcal{F}^{r,\ell_{\text{OT}}}_{\text{Leaky-2-Mult}}$ on input $(x_k^i \oplus \sigma^i_{k,v})_{k\in[r]}$ from P_i and $(\Delta_k[v])_{k\in[r]}$ from P_j. P_i receives $a^i_{k,v}$ and P_j receives $b^i_{k,v}$ such that $a^i_{k,v} \oplus b^i_{k,v} = \Delta_k[v] \cdot (x_k^i \oplus \sigma_k^i)$.
2. Each P_i, for $i \neq j$, outputs the ℓ_{BMR}-bit strings $Z_k^i := (a^i_{k,1} \oplus \rho^i_{k,1}, \ldots, a^i_{k,\ell_{\text{BMR}}} \oplus \rho^i_{k,\ell_{\text{BMR}}})$, for $k \in [r]$.
3. P_j outputs the ℓ_{BMR}-bit strings $Z_k^j := \bigoplus_{i\neq j}(b^i_{k,1}, \ldots, b^i_{k,\ell_{\text{BMR}}}) \oplus (\rho^j_{k,1}, \ldots, \rho^j_{k,\ell_{\text{BMR}}})$, for $k \in [r]$.

Fig. 11. n-party secret-shared bit/string multiplication using leaky 2-party multiplication

Communication complexity. The communication complexity of $\Pi^{r,\ell_{\text{BMR}}}_{\text{Bit}\times\text{String}}$ is exactly that of $(n-1)\ell_{\text{BMR}}$ instances of $\mathcal{F}^{r,\ell_{\text{OT}}}_{\text{Leaky-2-Mult}}$, where ℓ_{OT} is the leakage parameter used in the protocol $\Pi^{r,\ell_{\text{OT}}}_{\text{Leaky-2-Mult}}$. Note that ℓ_{OT} is independent of ℓ_{BMR} used in the bit/string protocol, but affects the security and cost of realising $\mathcal{F}_{\text{Leaky-2-Mult}}$. The total complexity is then $(n-1)\ell_{\text{BMR}}(\ell_{\text{OT}}(r+\kappa) + r)$ bits, or an amortized cost of $(n-1)\ell_{\text{BMR}}(\ell_{\text{OT}} + \ell_{\text{OT}}\kappa/r + 1)$ bits per multiplication.

Theorem 4.1 *Protocol $\Pi^{r,\ell_{\text{BMR}}}_{\text{Bit}\times\text{String}}$ UC-securely realizes $\mathcal{F}^{r,\ell_{\text{BMR}}}_{\text{BitString}}$ in the $\mathcal{F}^{2r}_{\text{Zero}}$-hybrid in the presence of static honest-but-curious adversaries, under the $\mathsf{DRSD}_{r,h,\ell_{\text{OT}}}$ assumption.*

The proof is a direct extension of the proof of Theorem 3.2.

4.3 The Preprocessing Protocol

Our protocol for generating the garbled circuit is shown in Figure 12. We use two functionalities $\mathcal{F}_{\text{Bit}\times\text{Bit}}$ (Figure 8) and $\mathcal{F}_{\text{BitString}}(P_j)$ (Figure 9) for multiplying two additively shared bits, and multiplying an additively shared bit with a string held by P_j, respectively. $\mathcal{F}_{\text{Bit}\times\text{Bit}}$ can be easily implemented using a multiplication triple from $\mathcal{F}_{\text{Triple}}$ in the previous section, whilst $\mathcal{F}_{\text{BitString}}$ uses a variant of the Π_{Triple} protocol optimized for this task.

The Preprocessing Protocol – $\Pi^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$

COMMON INPUT: $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^{\ell_{\text{BMR}}} \to \{0,1\}^{n\ell_{\text{BMR}}+1}$, a uniformly random sampled function and H' defined from H excluding the least significant bit of the output. A boolean circuit C_f with fan-out 1. Let $\mathsf{AND}, \mathsf{XOR}$ and SPLIT be the sets of AND, XOR and splitter gates, respectively. Given a gate, let I and O be the set of its input and output wires, respectively. If $g \in \mathsf{SPLIT}$, then $I = \{w\}$ and $O = \{u, v\}$, otherwise $O = \{w\}$.
For each $i \in [n]$, the protocol proceeds as follows:

1. **Free-XOR offsets:** For every $g \in \mathsf{XOR}$, P_i samples a random value $\Delta^i_g \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$
2. **Circuit-input wires' masks and keys:** If w is a *circuit-input* wire:
   (a) P_i samples a key $k^i_{w,0} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$ and a wire mask share $\lambda^i_w \leftarrow \{0,1\}$.
   (b) If w is input to a XOR gate g', P_i sets $k^i_{w,1} = k^i_{w,0} \oplus \Delta^i_{g'}$, otherwise $k^i_{w,1} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$.
3. **Intermediate wires' masks and keys:** Passing topologically through all the gates $g \in G = \{\mathsf{AND} \cup \mathsf{XOR} \cup \mathsf{SPLIT}\}$ of the circuit:
   (a) If $g \in \mathsf{XOR}$, P_i computes:
      – $\lambda^i_w = \bigoplus_{x\in I} \lambda^i_x$.
      – $k^i_{w,0} = \bigoplus_{x\in I} k^i_{x,0}$ and $k^i_{w,1} = k^i_{w,0} \oplus \Delta^i_g$.
   (b) If $g \notin \mathsf{XOR}$, P_i does as follows:
      – If $g \in \mathsf{AND}$, $\lambda^i_w \leftarrow \{0,1\}$. Else if $g \in \mathsf{SPLIT}$, sets $\lambda^i_x = \lambda^i_w$ for every $x \in O$.
      – For every $x \in O$, $k^i_{x,0} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$. If $x \in O$ is input to a XOR gate g', set $k^i_{x,1} = k^i_{x,0} \oplus \Delta^i_{g'}$, otherwise sample $k^i_{x,1} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$.
4. **Garble gates:** For each gate $g \in \{\mathsf{AND} \cup \mathsf{SPLIT}\}$, the parties run the subprotocol $\Pi^{\ell_{\text{BMR}}}_{\text{GateGarbling}}$, obtaining back shares $\tilde{g}^i$ of each garbled gate.
5. **Reveal input/output wires' masks:** For every *circuit-output* wire w, P_i broadcasts λ^i_w. For every *circuit-input* wire w, P_i sends λ^i_w to the party P_j who provides input on it. Each party reconstructs the wire masks from her received values as $\lambda_w = \bigoplus_{i=1}^n \lambda^i_w$.
6. **Open Garbling** For each $g \in \{\mathsf{AND} \cup \mathsf{SPLIT}\}$, P_i sends $\tilde{g}^i$ to P_1. P_1 reconstructs every garbled gate, $\tilde{g} = \bigoplus_{i=1}^n \tilde{g}^i$, and broadcasts it.

Fig. 12. The preprocessing protocol that realizes $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$

Most of the preprocessing protocol is similar to previous works [BLO16, HSS17], where first each party samples their sets of wire keys and shares of wire masks, and then the parties interact to obtain shares of the garbled gates. It is the second stage where our protocol differs, so we focus here on the details of the gate garbling procedures.

The Gate Garbling Protocol We describe the details of the sub-protocol $\Pi^{\ell_{\text{BMR}}}_{\text{GateGarbling}}$ (Figure 13), implementing the gate garbling phase of $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$. Creating garbled AND gates is done similarly to the OT-based protocol [BLO16], with the exception that we use short wire keys of length ℓ_{BMR} instead of κ. We also show how to create sharings of garbled splitter gates *without any interaction*, so these are much cheaper than AND gates.

The Gate Garbling sub-protocol – $\Pi^{\ell_{\text{BMR}}}_{\text{GateGarbling}}$

COMMON INPUT: a function $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^{\ell_{\text{BMR}}} \to \{0,1\}^{n\ell_{\text{BMR}}+1}$; H' defined as H excluding the least significant output bit; the gate g to be garbled.
PRIVATE INPUT: Each P_i, $i \in [n]$, holds λ^i_v and $k^i_{v,0}, k^i_{v,1}$, for each wire v.

1. If $g \in \mathsf{AND}$ with input wires $\{u, v\}$ and output wire w:
   (a) Each party P_i defines $R^i_g = k^i_{w,0} \oplus k^i_{w,1}$, for each $i \in [n]$
   (b) Call $\mathcal{F}_{\text{Bit}\times\text{Bit}}$ to compute shares of $\lambda_u \cdot \lambda_v$, and use these to locally obtain shares of
   $$\chi_{g,a,b} = (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w, \quad \text{for } (a,b) \in \{0,1\}^2$$
   (c) Call $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{BitString}}(P_i)$ to get shares of $\chi_{g,a,b} \cdot R^i_g$, for each $i \in [n]$ and $(a,b) \in \{0,1\}^2$. P_i then sets $\rho^i_{i,a,b} = k^i_{w,0} \oplus (\chi_{g,a,b} \cdot R^i_g)^i$, and $\forall j \neq i$, P_j sets $\rho^j_{i,a,b} = (\chi_{g,a,b} \cdot R^i_g)^j$.
   (d) Each P_i sets $\tilde{g}^i_{a,b} = \mathsf{H}(i,b,k^i_{u,a}) \oplus \mathsf{H}(i,a,k^i_{v,b}) \oplus (\chi^i_{g,a,b}, \rho^i_{1,a,b}, \ldots, \rho^i_{n,a,b})$, for $a, b \in \{0,1\}$.
2. If $g \in \mathsf{SPLIT}$ with input wire w and output wires $\{u, v\}$:
   (a) Call $\mathcal{F}^{2n\ell_{\text{BMR}}}_{\text{Zero}}$ twice, so that each P_i receives shares $s^i_0, s^i_1 \in \{0,1\}^{2n\ell_{\text{BMR}}}$.
   (b) P_i sets $\rho^i_c = s^i_c \oplus (0, \ldots, k^i_{u,c}, 0, \ldots, k^i_{v,c}, \ldots, 0)$ for $c \in \{0,1\}$.
   (c) Set $\tilde{g}^i_c = \big(\mathsf{H}'(i,0,k^i_{w,c}), \mathsf{H}'(i,1,k^i_{w,c})\big) \oplus \rho^i_c$, for $c \in \{0,1\}$.

Fig. 13. The gate garbling sub-protocol

Suppose that for an AND gate g, each P_i holds the wire mask share λ^i_v and keys $k^i_{v,0}, k^i_{v,1} \leftarrow \{0,1\}^{\ell_{\text{BMR}}}$. P_i defines $R^i_g = k^i_{w,0} \oplus k^i_{w,1}$. After that all parties call $\mathcal{F}_{\text{Bit}\times\text{Bit}}$ once to compute additive shares of $\lambda_{uv} = \lambda_u \cdot \lambda_v \in \{0,1\}$, which are then used to locally compute shares of $\chi_{g,a,b} = (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w$, for each $(a,b) \in \{0,1\}^2$. Each P_i obtains $\chi^i_{g,a,b}$ such that $\chi_{g,a,b} = \oplus_{i\in[n]} \chi^i_{g,a,b}$. To compute shares of the products $\chi_{g,a,b} \cdot R^i_g$, the parties call $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{BitString}}(P_i)$ three times, for each $i \in [n]$, to multiply R^i_g with each of the bits $\lambda_u, \lambda_v, (\lambda_{uv} \oplus \lambda_w)$. These can then be used for each P_j to locally obtain the shares $(\chi_{g,a,b} \cdot R^i_g)^j$, for all $(a,b) \in \{0,1\}^2$ (just as in [BLO16]).

After computing the bit/string products, P_j then computes for each $(a,b) \in \{0,1\}^2$:

$$\rho^j_{i,a,b} = \begin{cases} (\chi_{g,a,b} \cdot R^i_g)^j & j \neq i \\ k^i_{w,0} \oplus (\chi_{g,a,b} \cdot R^i_g)^i & j = i. \end{cases}$$

These values define shares of $\chi_{g,a,b} \cdot R^i_g \oplus k^i_{w,0}$. Finally, each party's share of the garbled AND gate is obtained as:

$$\tilde{g}^i_{a,b} = \mathsf{H}(i,b,k^i_{u,a}) \oplus \mathsf{H}(i,a,k^i_{v,b}) \oplus (\chi^i_{g,a,b}, \rho^i_{1,a,b}, \ldots, \rho^i_{n,a,b}), \quad a,b \in \{0,1\}$$

Summing up these values we obtain:

$$\begin{aligned} \bigoplus_i \tilde{g}^i_{a,b} &= \bigoplus_i \mathsf{H}(i,b,k^i_{u,a}) \oplus \mathsf{H}(i,a,k^i_{v,b}) \oplus (\chi^i_{g,a,b}, \rho^i_{1,a,b}, \ldots, \rho^i_{n,a,b}) \\ &= \bigoplus_{i=1}^{n} (\mathsf{H}(i,b,k^i_{u,a}) \oplus \mathsf{H}(i,a,k^i_{v,b})) \oplus (c, k^1_{w,c}, \ldots, k^n_{w,c}), \end{aligned}$$

where $c = \chi_{g,a,b}$, as required.

To garble a splitter gate, we observe that here there is no need for any secure multiplications within MPC, and the parties can produce shares of the garbled gate *without any interaction*. This is because the

two output wire values are the same as the input wire value, so to obtain a share of the encryption of the two output keys on wires u, v with input wire w, party P_i just computes:

$$(\mathsf{H}'(i, 0, k^i_{w,c}), \mathsf{H}'(i, 1, k^i_{w,c})) \oplus (0, \ldots, k^i_{u,c}, 0, \ldots, k^i_{v,c}, 0, \ldots, 0)$$

for $c \in \{0, 1\}$, where the right-hand vector contains P_i's keys in positions i and $n + i$. The parties then re-randomize this sharing with a share of zero from $\mathcal{F}_{\text{Zero}}$, so that opening the shares does not leak information on the individual keys.[8]

4.4 Security and Complexity

The above approach reduces size of the garbled circuit by a factor κ/ℓ_{BMR}, for ℓ_{BMR}-bit keys, but still requires n keys for every row in the garbled gates. Similarly to Section 3, when n is large we can reduce this by using a (random) committee $\mathcal{P}_{(h)}$ of size n_h that has at least h honest parties. $\Pi^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$ and $\Pi^{\ell_{\text{BMR}}}_{\text{BMR}}$ are then run as if called only by the parties in $\mathcal{P}_{(h)}$. For circuit-input wires w where parties in $\mathcal{P} \setminus \mathcal{P}_{(h)}$ provide input, they are sent the masks λ_w in $\Pi^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$, so in $\Pi^{\ell_{\text{BMR}}}_{\text{BMR}}$ (Figure 14) they can then broadcast $\Lambda_w = \rho^i_w \oplus \lambda_w$ in the same way as parties in $\mathcal{P}_{(h)}$.

This reduces the size of the garbled circuit by an additional factor of n/n_h. Finally, the same committee $\mathcal{P}_{(h)}$ can be combined with a (random) committee $\mathcal{P}_{(1)}$ with a single honest party in order to optimize the bit multiplications needed to compute the $\chi_{g,a,b}$ values, as was described in Section 3.

In Section 5, we give some examples of committee sizes and key lengths that ensure security, and compare this with the naive approach of running the preprocessing phase of BMR in $\mathcal{P}_{(1)}$ only.

Theorem 4.2 *Protocol $\Pi^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$ UC-securely realizes the functionality $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$ with perfect security in the $(\mathcal{F}_{\text{Bit}\times\text{Bit}}, \mathcal{F}^{\ell_{\text{BMR}}}_{\text{BitString}}, \mathcal{F}^{2n\ell_{\text{BMR}}}_{\text{Zero}})$-hybrid model in the presence of static honest-but-curious adversaries.*

Proof. Let $\mathcal{A}$ denote a PPT adversary corrupting a subset of parties $A \subset [n]$, then we prove that there exists a PPT simulator $\mathcal{S}$ that simulates the adversary's view. In the following, we denote by $\bar{A}$ the set of honest parties. When we say that the simulator is given some value, we mean that it receives it from $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$.

The description of the simulation: Denote by W and O_{C_f}, respectively, the set of wires and the set of circuit-output wires of a boolean circuit C_f. Denote by $I_{C_f,S}$ the set of circuit-input wires of a circuit where a subset of parties $S \subset [n]$ provides input to the circuit. We assume w.l.o.g. that $\mathcal{A}$ is a deterministic adversary, which receives as additional input a random tape that determines its internal coin tosses. Upon receiving $\mathcal{A}$'s input $(1^\kappa, A, C_f)$ and output $(\{\lambda_w\}_{w\in O_{C_f}}, \{k^i_{v,0}, k^i_{v,1}\}_{v\in W}, \{\lambda_u\}_{u\in I_{C_f,A}})$, $\mathcal{S}$ incorporates $\mathcal{A}$ and internally emulates an execution of the honest parties running $\Pi^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$ with the adversary $\mathcal{A}$.

1. CIRCUIT-INPUT WIRES' MASKS AND KEYS: For every *circuit-input* wire u and for $j \in A$, $\mathcal{S}$ samples from P_j's random tape the wire mask shares λ^j_u and the keys $k^j_{u,0}, k^j_{u,1}$ that party is meant to obtain from $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$. If a corrupted P_j provides input to the circuit on a given wire u, $\mathcal{S}$ samples $\{\lambda^i_u\}_{i\notin A}$ such that $\bigoplus_{i\notin A} \lambda^i_u = \lambda_u \oplus \bigoplus_{j\in A} \lambda^j_u$, where the value λ_u was received from $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$. If it is a honest party providing input on u, $\mathcal{S}$ samples $\{\lambda^i_u\}_{i\notin A}$ uniformly at random.

[8] For AND gates, the shares output by $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{BitString}}$ are uniformly random, so do not need re-randomizing with sharings of zero.

2. INTERMEDIATE WIRES' MASKS AND KEYS: Passing topologically through the gates g of the circuit:
   - For $j \in A$: If $g \in \mathsf{AND}$, $\mathcal{S}$ samples $\lambda_w^j \in \{0,1\}$ from P_j's random tape. If $g \in \mathsf{SPLIT}$, it sets $\lambda_x^j = \lambda_w^j$ for both output wires $x = u, v$. If $g \in \mathsf{XOR}$, it sets $\lambda_w^j = \bigoplus_{x \in I} \lambda_x^j$.
   - For $i \notin A$: If $g \in \mathsf{AND}$, $\mathcal{S}$ samples λ_w^i. If $g \in \mathsf{SPLIT}$, it sets $\lambda_x^i = \lambda_w^i$ for both output wires $x = u, v$. If $g \in \mathsf{XOR}$, it sets $\lambda_w^i = \bigoplus_{x \in I} \lambda_x^i$.
   - If x is a *circuit-output* wire, the simulator adjusts the value $\lambda_x \in \{0,1\}$ that $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$ sends to the parties to be $\lambda_x = \bigoplus_{i \notin A} \lambda_x^i \oplus \bigoplus_{j \in A} \lambda_x^j$.
3. GARBLE GATES: For each $g \in \mathsf{AND} \cup \mathsf{SPLIT}$:
   - If $g \in \mathsf{AND}$, let u_g, v_g be its input wires and w_g its output wire. $\mathcal{S}$ emulates $\mathcal{F}_{\text{Bit}\times\text{Bit}}$ by sampling shares z_g^j from P_j's random tape, for $j \in A$, and setting random z_g^i for $i \notin A$ such that $\sum_{i \in [n]} z_g^i = \lambda_{u_g} \cdot \lambda_{v_g}$, where $\lambda_{u_g}, \lambda_{v_g}$ were obtained from $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$. $\mathcal{S}$ has now all the values to compute shares of $\chi_{g,a,b}$ as $\chi_{g,a,b}^i = a \cdot b \oplus b \cdot \lambda_{u_g}^i \oplus a \cdot \lambda_{v_g}^i \oplus z_g^i \oplus \lambda_{w_g}^i$ for $i \in [n]$.
     For $j \in [n]$, $\mathcal{S}$ emulates three calls to $\mathcal{F}_{\text{BitString}}^{\ell_{\text{BMR}}}(P_j)$ with inputs $\{\chi_{g,0,0}^i, \chi_{g,0,1}^i, \chi_{g,1,0}^i\}$ from every P_i and additional input R_g^j from P_j, where $R_g^j = k_{w_g,0}^j \oplus k_{w_g,1}^j$. In each of these emulated calls and for $i \in A$, it computes the corrupted parties' output shares from P_i's random tape, while for $i \notin A$ it samples random shares that sum to each of the values $R_g^j \cdot \chi_{g,0,0}$, $R_g^j \cdot \chi_{g,0,1}$ and $R_g^j \cdot \chi_{g,1,0}$ as required.
   - If $g \in \mathsf{SPLIT}$, $\mathcal{S}$ emulates twice $\mathcal{F}_{\text{Zero}}^{2n\ell_{\text{BMR}}}$ by computing shares s_0^i, s_1^i from P_i's random tape for $i \in A$ and setting s_0^j, s_1^j for $j \notin A$ such that $\bigoplus_{i \in [n]} s_c^i = 0$ for $c \in \{0,1\}$.

   Setting the ρ and $\tilde{g}$ values is local computation.
4. REVEAL INPUT/OUTPUT WIRES' MASKS: For every *circuit-output* wire w, $\mathcal{S}$ adds values λ_w^i, $i \notin A$ (previously computed in Step 2) to the view of each P_j, $j \in A$. For every *circuit-input* wire u on which a P_j, $j \in A$, provides input, $\mathcal{S}$ adds the $\{\lambda_u^i\}_{i \notin A}$ values it previously computed in Step 1 to P_j's view.
5. OPEN GARBLING: Using the adversary's output $\{\tilde{g}\}_{g \in \mathsf{AND} \cup \mathsf{SPLIT}}$, $\mathcal{S}$ proceeds as follows: If $1 \in A$, it plays the role of each P_j, for $j \notin A$, and sends to P_1 the shares $\{\tilde{g}^j\}_{g \in \mathsf{AND} \cup \mathsf{SPLIT}}$ that it previously computed. Otherwise, the simulator plays the role of P_1 by sending $\{\tilde{g}\}_{g \in \mathsf{AND} \cup \mathsf{SPLIT}}$ to each $P_i, i \in A$.

Indistinguishability: The wire keys and the (circuit-input and circuit-output) wire masks output by the functionality $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$ are i.i.d. uniformly random variables in the real world too. In both worlds and for the additional simulated values, the corrupted parties' shares for the wire masks, the bit products ($\mathcal{F}_{\text{Bit}\times\text{Bit}}$ functionality) and bit/string products ($\mathcal{F}_{\text{BitString}}^{\ell_{\text{BMR}}}$ functionality) needed to garble AND gates are fixed by $\mathcal{A}$'s random tape, while the honest parties' shares of the same values are uniformly random additive shares. In particular, this implies that shares $\tilde{g}_{a,b}^i$ of garbled AND gates are uniformly random additive shares in both executions. The same applies to shares of garbled splitter gates, due to the use of the $\mathcal{F}_{\text{Zero}}^{2n\ell_{\text{BMR}}}$ functionality in the real world. Regarding the OPEN GARBLING step, if $1 \notin A$ the reconstructed garbled circuit is identically distributed in both worlds. Else, if $1 \in A$, the adversary gets additive shares of the garbled circuit both in the real and simulated executions, as we argued.

Finally, the distribution of the variables corresponding to additive shares, on the one hand, and that of the i.i.d. variables, on the other hand, guarantees that the joint output of all parties, together with the simulated/real view of corrupted parties, are identically distributed in both worlds. More formally, let $\texttt{output}^{\pi}(\boldsymbol{x}, \kappa)$ (resp. $f(\boldsymbol{x})$) be the output of $\Pi_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$ (resp. $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$) on input $\boldsymbol{x} \in \{0,1\}^*$ from all parties and security parameter κ. Let $\texttt{view}_A^{\pi}(\boldsymbol{x}, \kappa)$ (resp. $f_A(\boldsymbol{x})$) be the restriction of these outputs to the set of corrupted parties A. We just proved that:

$$\{(\mathcal{S}(1^{\kappa}, \boldsymbol{x}, f_A(\boldsymbol{x})), f(\boldsymbol{x}))\}_{\boldsymbol{x},\kappa,A} \approx \{(\texttt{view}_A^{\pi}(\boldsymbol{x}, \kappa), \texttt{output}^{\pi}(\boldsymbol{x}, \kappa))\}_{\boldsymbol{x},\kappa,A}.$$

□

4.5 The Online Phase

Protocol $\Pi^{\ell_{\mathrm{BMR}}}_{\mathrm{BMR}}$

COMMON INPUT: A boolean circuit C_f with fan-out-one gates representing the function f. Let $\mathsf{AND}, \mathsf{XOR}$ and SPLIT be the sets of AND, XOR and splitter gates, respectively. For a gate $g \in \mathsf{SPLIT}$, let the input wire be $\{w\}$ and the output wires be $\{u, v\}$. Otherwise let $\{u, v\}$ be the input wires and $\{w\}$ the output wire.

CRS: $\mathsf{H} : [n] \times \{0,1\} \times \{0,1\}^{\ell_{\mathrm{BMR}}} \to \{0,1\}^{n\ell_{\mathrm{BMR}}+1}$, a uniformly random function, and H' defined from H by excluding the least significant bit of the output.
The parties execute the following commands in sequence:

Preprocessing:

1. Call $\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{Preprocessing}}$ with input C_f. Each party P_i obtains the garbled version $\tilde{g}$ of every gate $g \in \mathsf{SPLIT} \cup \mathsf{AND}$, the wire masks λ_w for every output wire and every wire associated with their input, and all their keys $\{k^i_{w,0}, k^i_{w,1}\}$ for every wire w of the circuit.

Online Computation:

1. For all input wires w with input from P_i, party P_i computes $\Lambda_w = \rho^i_w \oplus \lambda_w$, where ρ^i_w is P_i's input to C_f on wire w, and λ_w was obtained from $\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{Preprocessing}}$. Then, P_i broadcasts the public value Λ_w to all parties.
2. For all input wires w, each party P_i broadcasts the key k^i_{w,Λ_w} associated to Λ_w.
3. Passing through the circuit topologically, the parties can now locally compute the following operations for each gate g.
   (a) If $g \in \mathsf{SPLIT}$, set $\Lambda_x = \Lambda_w$ for $x \in \{u, v\}$ and then compute:

$$(k^1_{u,\Lambda_u}, \ldots, k^n_{u,\Lambda_u}, k^1_{v,\Lambda_v}, \ldots, k^n_{v,\Lambda_v}) = \tilde{g}_{\Lambda_w} \oplus \left(\bigoplus_{i=1}^{n} \mathsf{H}'(i, 0, k^i_{w,\Lambda_w}), \bigoplus_{i=1}^{n} \mathsf{H}'(i, 1, k^i_{w,\Lambda_w}) \right)$$

   (b) If $g \in \mathsf{AND}$, the parties compute:

$$(\Lambda_w, k^1_{w,\Lambda_w}, \cdots, k^n_{w,\Lambda_w}) = \tilde{g}_{\Lambda_u,\Lambda_v} \oplus \bigoplus_{i=1}^{n} \Big(\mathsf{H}(i, \Lambda_v, k^i_{u,\Lambda_u}) \oplus \mathsf{H}(i, \Lambda_u, k^i_{v,\Lambda_v}) \Big)$$

   (c) If $g \in \mathsf{XOR}$, the parties compute $\Lambda_w = \bigoplus_{x \in I} \Lambda_x$ and $k^i_{w,\Lambda_w} = \bigoplus_{x \in I} k^i_{x,\Lambda_x}$ for $i \in [n]$.
4. Eventually, all parties will obtain a public value Λ_w for every circuit-output wire w. The party can then recover the actual output value from $\rho_w = \Lambda_w \oplus \lambda_w$, where λ_w was obtained in the preprocessing stage.

Fig. 14. Online phase of the constant-round MPC protocol

We present the online phase of our protocol for multi-party garbled circuits with short keys in Figure 14 . Given the previous description of the garbling phase, the online phase is quite straightforward, where upon reconstructing the garbled circuit and obtaining all input keys, the evaluation process is similar to [BMR90]. As in that work, all parties run the evaluation algorithm, which in our case involves each party computing just $2n$ hash evaluations per gate. During evaluation, the parties only see the randomly masked wire values, which we call "public values", and cannot determine the actual values being computed. Upon completion, the parties obtain the actual output using the output wire masks revealed from $\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{Preprocessing}}$. The security of the protocol reduces to the $\mathsf{DRSD}_{r,h,\ell_{\mathrm{BMR}}}$ problem, where ℓ_{BMR} is the key length, h is the number of honest parties, and r is twice the output length of the function H (sampled by the CRS).

We remark that in practice, we may want to implement the random function H in the CRS using fixed-key AES in the ideal cipher model, as is common for garbling schemes based on free-XOR. In Appendix C.2, we show that this reduces the number of AES calls from $O(n^2)$ in previous BMR protocols to $O(n^2\ell_{\text{BMR}}/\kappa)$.

We conclude with the following theorem.

Theorem 4.3 *Let f be an n-party functionality $\{0,1\}^{n\kappa} \mapsto \{0,1\}^{\kappa}$ and assume that the $\mathsf{DRSD}_{2r,h,\ell_{\text{BMR}}}$ assumption (cf. Definition 2.4) holds, where $r = n\ell_{\text{BMR}} + 1$. Then Protocol $\Pi_{\text{BMR}}^{\ell_{\text{BMR}}}$ from Figure 14 UC-securely computes f in the presence of a static honest-but-curious adversary corrupting $t = n - h$ parties in the $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$-hybrid model.*

Proof. We reduce security of the protocol to the extended double-key decisional-RSD problem (Definition 2.7) with parameters (r, h, ℓ), where $r := n\ell_{\text{BMR}} + 1$. By Lemma 2.5, this is reducible to $\mathsf{DRSD}_{2r,h,\ell}$.

Let $\mathcal{A}$ be a PPT adversary corrupting a subset of parties $A \subset [n]$ such that $|A| = n - h$. We prove that there exists a PPT simulator $\mathcal{S}$, with access to an ideal functionality $\mathcal{F}$ that implements f, which simulates the adversary's view. The simulator fixes the CRS as a random $2n \cdot 2^{\ell_{\text{BMR}}} \times 2^{n\ell_{\text{BMR}}+1}$ matrix. A key k_w for wire w is denoted as an active key if it is observed by the adversary upon evaluating the garbled circuit. The remaining hidden key is denoted as an inactive key. An active path is the set of all active keys that are observed throughout the garbled circuit evaluation.

Denoting the set of honest parties by $\bar{A}$, our simulator $\mathcal{S}$ is defined below.

The description of the simulation.

1. INITIALIZATION. Upon receiving the adversary's input $(1^\kappa, A, \boldsymbol{x}_A)$ and output $\boldsymbol{y}$, $\mathcal{S}$ samples a i.i.d uniformly random tapes r_i for each $i \in A$, incorporates $\mathcal{A}$ and internally emulates an execution of the honest parties running $\Pi_{\text{BMR}}^{\ell_{\text{BMR}}}$ with the adversary $\mathcal{A}$. When we say that $\mathcal{S}$ chooses a value for some corrupted party, we mean that it samples the value from that party's random tape r_i.
2. PREPROCESSING. $\mathcal{S}$ obtains the adversary's input C_f which is a Boolean circuit that computes f with a set of wires W and a set of G gates, and emulates $\mathcal{F}_{\text{Preprocessing}}^{\ell_{\text{BMR}}}$, as follows:
   - For every XOR gate g and $i \in A$ the simulator samples $\Delta_g^i \in \{0,1\}^{\ell_{\text{BMR}}}$.
   - For every input wire u the simulator chooses a random bit $\Lambda_u \in \{0,1\}$ and, for every $i \in \bar{A}$, an active key $k_{u,\Lambda_u}^i \in \{0,1\}^{\ell_{\text{BMR}}}$. Additionally, it chooses a key $k_{u,0}^i \in \{0,1\}^{\ell_{\text{BMR}}}$ for every $i \in A$. Finally, and also for $i \in A$, if u is input to a XOR gate g' it sets $k_{u,1}^i = k_{u,0}^i \oplus \Delta_{g'}^i$, otherwise it samples $k_{u,1}^i \in \{0,1\}^{\ell_{\text{BMR}}}$.

   The simulator continues the emulation of the garbling phase by computing an active path of the garbled circuit that corresponds to the sequence of keys which will be observed by the adversary. Importantly, $\mathcal{S}$ never samples the inactive keys $k_{u,\bar{\Lambda}_u}^i$, $k_{v,\bar{\Lambda}_v}^i$ and $k_{w,\bar{\Lambda}_w}^i$ for $i \in \bar{A}$ in order to generate the garbled circuit.
   - ACTIVE PATH GENERATION OF XOR GATES. For every XOR gate g with input a set of wires I and an output wire w,
     - $\mathcal{S}$ sets $\Lambda_w = \bigoplus_{x \in I} \Lambda_x$.
     - Next, for $i \in A$ it sets $k_{w,0}^i = \bigoplus_{x \in I} k_{x,0}^i$ and $k_{w,1}^i = k_{w,0}^i \oplus \Delta_g^i$.
     - Finally, for $i \in \bar{A}$ the simulator sets $k_{w,\Lambda_w}^i = \bigoplus_{x \in I} k_{x,\Lambda_x}^i$.

– ACTIVE PATH GENERATION OF AND GATES. For every AND gate g with input wires $I = \{u, v\}$ and an output wire w, $\mathcal{S}$ samples a random $\Lambda_w \in \{0,1\}$ and honestly generates the entry in row (Λ_u, Λ_v), where Λ_u (resp. Λ_v) is the public value associated to the left (resp. right) input wire to g. Namely, the simulator computes

$$\tilde{g}_{\Lambda_u,\Lambda_v} = \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, \Lambda_v, k^i_{u,\Lambda_u}) \oplus \mathsf{H}(i, \Lambda_u, k^i_{v,\Lambda_v}) \right) \oplus (\Lambda_w, k^1_{w,\Lambda_w}, \ldots, k^n_{w,\Lambda_w}).$$

The remaining three rows are sampled uniformly at random from $\{0,1\}^{n\ell_{\mathrm{BMR}}+1}$.

– ACTIVE PATH GENERATION OF SPLITTER GATES. For every splitter gate g with an input wire $I = \{w\}$ and output wires $O = \{u, v\}$, $\mathcal{S}$ sets $\Lambda_x = \Lambda_w$ for every $x \in O$ and honestly generates the entry in row Λ_w, where Λ_w is the public value associated to the input wire to g. Namely, the simulator computes

$$\tilde{g}_{\Lambda_w} = \left(\bigoplus_{i=1}^{n} \mathsf{H}'(i, 0, k^i_{w,\Lambda_w}), \bigoplus_{i=1}^{n} \mathsf{H}'(i, 1, k^i_{w,\Lambda_w}) \right) \oplus (k^1_{u,\Lambda_u}, \ldots, k^n_{u,\Lambda_u}, k^1_{v,\Lambda_v}, \ldots, k^n_{v,\Lambda_v}).$$

The remaining row is sampled uniformly at random from $\{0,1\}^{2n\ell_{\mathrm{BMR}}}$.

– SETTING THE TRANSLATION TABLE. For every output wire $w \in W$ returning the ith bit of $\boldsymbol{y}$, the simulator sets $\lambda_w = \Lambda_w \oplus y_i$. For all input wires $w \in W''$ that are associated with the ith bit of $\boldsymbol{x}_A$ (the adversary's input), the simulator sets $\lambda_w = \Lambda_w \oplus \boldsymbol{x}_{A,i}$. The simulator forwards the adversary the λ_w value for every output wire $w \in W$ and every circuit-input wire $w \in W''$ associated with a corrupted party. It completes the emulation of $\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{Preprocessing}}$ by adding the complete garbled circuit to the view of each corrupted party.

3. ONLINE COMPUTATION. In the online computation the simulator adds to the view of every corrupted partiy the public values $\{\Lambda_w\}_{w \in W'}$ that are associated with the honest parties' input wires W'. The simulator adds the honest parties' input keys $\{k^i_{w,\Lambda_w}\}_{i \in \bar{A}, w \in W'}$ to the view of each corrupted party.

This concludes the description of the simulation. Note that the difference between the simulated and the real executions is regarding the way the garbled circuit is generated. More concretely, the simulated garbled gates include a single row that is properly produced, whereas the remaining three rows are picked at random. Let $\mathbf{HYB}^{\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{Preprocessing}}}_{\Pi^{\ell_{\mathrm{BMR}}}_{\mathrm{BMR}},\mathcal{A},\mathcal{Z}}(1^\kappa, z)$ denote the output distribution of the adversary $\mathcal{A}$ and honest parties in a real execution using $\Pi^{\ell_{\mathrm{BMR}}}_{\mathrm{BMR}}$ with adversary $\mathcal{A}$. Moreover, let $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)$ denote the output distribution of $\mathcal{S}$ and the honest parties in an ideal execution.

We prove that the ideal and real executions are indistinguishable.

Lemma 4.1 *The following two distributions are computationally indistinguishable:*

– $\{\mathbf{HYB}^{\mathcal{F}^{\ell_{\mathrm{BMR}}}_{\mathrm{Preprocessing}}}_{\Pi^{\ell_{\mathrm{BMR}}}_{\mathrm{BMR}},\mathcal{A},\mathcal{Z}}(1^\kappa, z)\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}$

– $\{\mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)\}_{\kappa \in \mathbb{N}, z \in \{0,1\}^*}$

Proof: We begin by defining a slightly modified simulated execution $\widetilde{\mathbf{HYB}}$, where the generation of the garbled circuit is modified so that upon receiving the parties' inputs $\{\rho^i\}_{i \in [n]}$ the simulator $\widetilde{\mathcal{S}}$ first evaluates the circuit C_f, computing the actual bit ρ_w to be transferred via wire w for all $w \in W$, where

W is the set of wires of C_f. It then chooses wire mask shares and wire keys as in the description of functionality $\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}$ from Figure 7. Finally, $\widetilde{\mathcal{S}}$ fixes the active key for each wire $w \in W$ to be $(k^1_{w,\rho_w \oplus \lambda_w}, \ldots, k^n_{w,\rho_w \oplus \lambda_w})$. The rest of this hybrid is identical to the simulation. This hybrid execution is needed in order to construct a distinguisher for the Extended Double-Key RSD assumption.

Let $\widetilde{\mathbf{HYB}}^{\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}}_{\Pi^{\ell_{\text{BMR}}}_{\text{BMR}},\mathcal{A}}(1^\kappa, z)$ denote the output distribution of the adversary $\mathcal{A}$ and honest parties in this game. It is simple to verify that the adversary's views in $\widetilde{\mathbf{HYB}}$ and $\mathbf{IDEAL}$ are identical, as in both cases the garbling of each gate includes just a single row that is correctly garbled and the public value associated with each wire w is independent of $\ell_{\text{BMR}w}$.

Our proof of the lemma follows by a reduction to the Extended Double-Key RSD hardness assumption (cf. Definition 2.7). Assume by contradiction the existence of an environment $\mathcal{Z}$, an adversary $\mathcal{A}$ and a non-negligible function $p(\cdot)$ such that

$$\left| \Pr[\mathcal{Z}(\mathbf{HYB}^{\mathcal{F}^{\ell_{\text{BMR}}}_{\text{Preprocessing}}}_{\Pi^{\ell_{\text{BMR}}}_{\text{BMR}},\mathcal{A},\mathcal{Z}}(1^\kappa, z)) = 1] - \Pr[\mathcal{Z}(\widetilde{\mathbf{HYB}}_{\Pi^{\ell_{\text{BMR}}}_{\text{BMR}},\mathcal{A},\mathcal{Z}}(1^\kappa, z)) = 1] \right| \geq \frac{1}{p(\kappa)}$$

for infinitely many κ's where the probability is taken over the randomness of $\mathcal{Z}$ as well as the randomness for choosing the Λ values and the keys. Then we construct a PPT distinguisher $\mathcal{D}$ for the Extended Double-Key RSD assumption that distinguishes between an instance of the form

$$\left(\mathsf{H}, \bigoplus_{i \in \bar{A}} \mathsf{H}(i,0,k_i), \bigoplus_{i \in \bar{A}} \mathsf{H}(i,0,k'_i), \bigoplus_{i \in \bar{A}} \mathsf{H}(i,1,k_i), \bigoplus_{i \in \bar{A}} \mathsf{H}(i,1,k'_i) \right)$$

and five random elements, for some subset $\bar{A}$ of $[n]$ of size h (that corresponds to the set of honest parties) with probability at least $\frac{1}{p(\kappa)\cdot|C|}$ via a sequence of hybrid games $\{\mathbf{HYB}_i\}_{i\in[|C|]}$, where $C = \mathsf{SPLIT} \cup \mathsf{AND}$. In more details, we define hybrid $\mathbf{HYB}_i$ as a hybrid execution with a simulator $\mathcal{S}_i$ that garbles the circuit as follows. The first i gates in the topological order are garbled as in the simulation whereas the remaining $|C| - i$ gates are garbled as in the real execution. Note that $\mathbf{HYB}_0$ is distributed as hybrid $\mathbf{HYB}$ and that $\mathbf{HYB}_{|C|}$ is distributed as $\widetilde{\mathbf{HYB}}$. Therefore, if $\mathbf{HYB}$ and $\widetilde{\mathbf{HYB}}$ are distinguishable with probability $\frac{1}{p(\kappa)}$ then there exists $\tau \in [|C|]$ such that hybrids $\mathbf{HYB}_{\tau-1}$ and $\mathbf{HYB}_\tau$ are distinguishable with probability at least $\frac{1}{p(\kappa)\cdot|C|}$. Next, we formally describe our reduction to the Extended Double-Key RSD hardness assumption. Upon receiving a tuple $(\mathsf{H}, \widetilde{\mathsf{H}}_0, \widetilde{\mathsf{H}}'_0, \widetilde{\mathsf{H}}_1, \widetilde{\mathsf{H}}'_1)$ that is distributed according to the first or the second distribution, a subset $\bar{A}$ of $[n]$ that denotes the set of honest parties, an index τ and the environment's input z, distinguisher $\mathcal{D}$ internally invokes $\mathcal{Z}$ and simulator $\mathcal{S}$. In more details,

- $\mathcal{D}$ internally invokes $\mathcal{Z}$ that fixes the honest parties' inputs $\boldsymbol{\rho}$.
- $\mathcal{D}$ emulates the communication with the adversary (controlled by $\mathcal{Z}$) in the initialization, preprocessing and garbling steps as in the simulation with $\mathcal{S}$.
- For each wire u, let $\rho_u \in \{0,1\}$ be the actual value on wire u. Note that these values, as well as the output of the computation y, can be determined since $\mathcal{D}$ knows the actual input of all parties to the circuit.
- For each wire u in the circuit and $i \in A$, $\mathcal{D}$ chooses a pair of keys $k^i_{u,0}, k^i_{u,1} \in \{0,1\}^{\ell_{\text{BMR}}}$, whereas for all $i \in \bar{A}$ it samples a random key $k^i_{u,\Lambda_u} \in \{0,1\}^{\ell_{\text{BMR}}}$. $\mathcal{D}$ further fixes the public value $\Lambda_u = \lambda_u \oplus \rho_u$.
- $\mathcal{D}$ then garbles the circuit as follows.

- For every $g_j \in \mathsf{AND}$ with input wires u and v and output wire w, $\mathcal{D}$ continues as follows.
  If $j < \tau$ then $\mathcal{D}$ garbles g_j exactly as in the simulation with $\widetilde{\mathcal{S}}$.
  If $j = \tau$ then $\mathcal{D}$ first honestly computes the (Λ_u, Λ_v)-th row by fixing

$$\tilde{g}_{\Lambda_u,\Lambda_v} = \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, \Lambda_v, k^i_{u,\Lambda_u}) \oplus \mathsf{H}(i, \Lambda_u, k^i_{v,\Lambda_v})\right) \oplus (c, k^1_{w,\Lambda_w}, \ldots, k^n_{w,\Lambda_w})$$

  where $c = \Lambda_w$.
  Next, $\mathcal{D}$ samples an inactive key $k^i_{w,\bar{\Lambda}_w}$ for all $i \in \bar{A}$ and fixes the remaining three rows as follows.

$$\begin{aligned}
\tilde{g}_{\Lambda_u,\bar{\Lambda}_v} &= \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, \bar{\Lambda}_v, k^i_{u,\Lambda_u}) \oplus \left(\bigoplus_{i\in A} \mathsf{H}(i, \Lambda_u, k^i_{v,\bar{\Lambda}_v})\right) \oplus \widetilde{\mathsf{H}}'_{\Lambda_u}\right) \\
&\quad \oplus (c, k^1_{w,c}, \ldots, k^n_{w,c}), \quad \text{where } c = \Lambda_u \cdot \bar{\Lambda}_v \oplus \Lambda_w \oplus \rho_w \\
\tilde{g}_{\bar{\Lambda}_u,\Lambda_v} &= \left(\bigoplus_{i\in A} \mathsf{H}(i, \Lambda_v, k^i_{u,\bar{\Lambda}_u}) \oplus \widetilde{\mathsf{H}}_{\Lambda_v} \oplus \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, \bar{\Lambda}_u, k^i_{v,\Lambda_v})\right)\right) \\
&\quad \oplus (c, k^1_{w,c}, \ldots, k^n_{w,c}), \quad \text{where } c = \bar{\Lambda}_u \cdot \Lambda_v \oplus \Lambda_w \oplus \rho_w \\
\tilde{g}_{\bar{\Lambda}_u,\bar{\Lambda}_v} &= \left(\bigoplus_{i\in A} \mathsf{H}(i, \bar{\Lambda}_v, k^i_{u,\bar{\Lambda}_u}) \oplus \widetilde{\mathsf{H}}_{\bar{\Lambda}_v} \oplus \left(\bigoplus_{i\in A} \mathsf{H}(i, \bar{\Lambda}_u, k^i_{v,\bar{\Lambda}_v})\right) \oplus \widetilde{\mathsf{H}}'_{\bar{\Lambda}_u}\right) \\
&\quad \oplus (c, k^1_{w,c}, \ldots, k^n_{w,c}), \quad \text{where } c = \bar{\Lambda}_u \cdot \bar{\Lambda}_v \oplus \Lambda_w \oplus \rho_w.
\end{aligned}$$

  Finally, if $j > \tau$ then $\mathcal{D}$ garbles g_j exactly as in hybrid $\mathbf{HYB}$. For that, $\mathcal{D}$ needs to know both active and inactive keys. It therefore chooses the inactive keys that are associated with the input and output wires of this gate for $i \in \bar{A}$, in order to be able to complete the garbling. Recall that the circuit is with fan-out 1. Therefore the distinguisher can choose the inactive key for the input wire of this gate (as it was not used as an input wire to gate g_τ).
- For every $g_j \in \mathsf{SPLIT}$ with input wire w and output wires u, v, $\mathcal{D}$ completes the garbling as follows.
  If $j < \tau$ then $\mathcal{D}$ garbles g_j exactly as in the simulation with $\widetilde{\mathcal{S}}$.
  If $j = \tau$ then $\mathcal{D}$ first honestly computes the Λ_wth row by fixing

$$\tilde{g}_{\Lambda_w} = \left(\bigoplus_{i=1}^{n} \mathsf{H}(i, 0, k^i_{w,\Lambda_w}) \oplus \mathsf{H}(i, 1, k^i_{w,\Lambda_w})\right) \oplus (k^1_{u,\Lambda_u}, \ldots, k^n_{v,\Lambda_v}).$$

  Next, it samples inactive keys $k^i_{u,\bar{\Lambda}_u}, k^i_{v,\bar{\Lambda}_v}$ for all $i \in \bar{A}$ and fixes the remaining row as follows.

$$\tilde{g}_{\bar{\Lambda}_w} = \left(\bigoplus_{i\in A} \mathsf{H}(i, 0, k^i_{w,\bar{\Lambda}_w}) \oplus \widetilde{\mathsf{H}}_0 \oplus \bigoplus_{i\in A} \mathsf{H}(i, 1, k^i_{w,\bar{\Lambda}_w}) \oplus \widetilde{\mathsf{H}}_1\right) \oplus (k^1_{u,\bar{\Lambda}_u}, \ldots, k^n_{v,\bar{\Lambda}_v}).$$

  If $j > \tau$ then $\mathcal{D}$ garbles g_j as in hybrid $\mathbf{HYB}$ using a similar process as for the case of an AND gate.

– This concludes the description of the reduction. Note that the set XOR need not be part of these hybrids since we do not send any garbling information for this set of gates. $\mathcal{D}$ hands the adversary the complete description of the garbled circuit and concludes the execution as in the simulation with $\widetilde{\mathcal{S}}$.
– $\mathcal{D}$ outputs whatever $\mathcal{Z}$ does.

Note first that if $(\widetilde{\mathsf{H}}_0, \widetilde{\mathsf{H}}'_0, \widetilde{\mathsf{H}}_1, \widetilde{\mathsf{H}}'_1)$ are truly uniform then the view generated by $\mathcal{D}$ is distributed as in $\mathbf{HYB}_\tau$. This is because only the active path is created as in the real execution, whereas the remaining rows are sampled uniformly at random from the appropriate domain. On the other hand, if this tuple is generated according to the following distribution

$$\left(\mathsf{H}, \bigoplus_{i\in A}\mathsf{H}(i,0,k_i), \bigoplus_{i\in A}\mathsf{H}(i,0,k'_i), \bigoplus_{i\in A}\mathsf{H}(i,1,k_i), \bigoplus_{i\in A}\mathsf{H}(i,1,k'_i)\right)$$

then this emulates game $\mathbf{HYB}_{\tau-1}$, since each tuple element emulates an evaluation of the hash values for the honest parties on the secret keys.

This completes the proof. □

5 Complexity Analysis and Implementation Results

We now compare the complexity of the most relevant aspects of our approach to the state-of-the-art prior results in semi-honest MPC protocols with dishonest majority. To demonstrate the practicality of our approach, we also present implementation results for the online evaluation phase of our BMR-based protocol.

5.1 Threshold Variants of Full-Threshold Protocols

Since the standard GMW and BMR-based protocols allow for up to $n-1$ corruptions, we also show how to modify previous protocols to support some threshold t, and compare our protocols with these variants. The method is very simple (and similar to the use of committees in our protocols), but does not seem to have been explicitly mentioned in previous literature. To evaluate a circuit C, all parties first secret-share their inputs to an arbitrarily chosen committee $\mathcal{P}'$, of size $t+1$. Committee $\mathcal{P}'$ runs the full-threshold protocol for a modified circuit C', which takes all the shares as input, and first XORs them together so that it computes the same function as C. The committee $\mathcal{P}'$ then sends the output to all parties in $\mathcal{P}$. The complexity of the threshold-t variant of a full-threshold protocol, Π, is then essentially the same as running Π between $t+1$ parties instead of n.

5.2 Concrete Hardness of RSD and Our Choice of Parameters

In this section we give an overview of how we select the key length ℓ in our protocols according to n, h, r, so that the corresponding $\mathsf{RSD}_{r,h,\ell}$ instance is hard enough. See Appendix B for a more detailed survey of known attacks and the techniques involved. As discussed in Section 2.3, RSD is similar to the (standard) syndrome decoding problem, where each component of the error vector is 0 or 1 with some constant probability, which is equivalent to the problem of *learning parity with noise* (LPN).

The most efficient attacks on RSD are Information Set Decoding (ISD), introduced by Prange in 1962 [Pra62], Wagner's Generalised Birthday Attack (GBA) [Wag02], and the Linearization Attack (LA) by Bellare et al. [BM97] and Saarinen [Saa07]. We stress that the goal of our analysis is to find a reasonable estimation of the complexity of these attacks; giving a complete description of all possible decoding techniques and a precise evaluation of their cost is out of the scope of this paper. In our analysis we intentionally underestimate the complexity of all the attacks, resulting in a conservative estimate of the security of our protocols.

When considering the hardness of RSD instances we need to distinguish the case where the solution to the problem is unique and the case of multiple solutions. In the first case, which always occurs for our BMR-style protocols, GBA essentially reduces to the classical birthday attack and the most efficient attack is ISD.

n	20			50				80				100				200		400	
h	10	11	16	10	20	25	40	16	24	32	56	20	30	40	60	60	100	80	120
ℓ Pra	19	18	13	21	13	11	8	32	12	10	8	14	11	9	8	8	8	8	8
ℓ	32	29	18	> 32	27	16	8	> 32	30	17	8	> 32	25	15	8	14	8	11	8

Table 1. Min key-length for BMR-style MPC with 128 bits of security for different n and h when $r = 2\ell n + 2$

h	15		20		30		40		50		80	
r	300	1500	300	3000	300	2000	400	3000	450	1000	420	2500
ℓ	14	26	11	32	8	16	7	15	6	8	4	8

Table 2. Min key-length for GMW-style MPC with 128 bits of security for different n and h

# parties n (honest) (ℓ_{OT}, r)	20 (6) $(31, 300)$	50 (15) $(14, 300)$	60 (20) $(11, 300)$	80 (30) $(8, 300)$	150 (40) $(7, 400)$	200 (50) $(6, 450)$	400 (120) $(1, 80)$
GMW ($t = n - 1$)	25.46	164.15	237.18	423.44	1497.5	2666.6	10693.2
GMW ($t = n - h$)	14.07	84.42	109.88	170.85	818.07	1517.55	5271.56
Ours	**12.89**	**37**	**40.38**	**50.01**	**169.36**	**261.6**	**403.63**

Table 3. Amortized communication cost (in kbit) of producing a single triple in GMW. We consider [DKS+17] for 1-out-of-4 OT extension in the GMW protocols, and the protocol from Section 3 in our work.

Classical information set decoding algorithms do not take into account the possibility that the solution is regular. In practice, when we estimate the cost of this attack, we consider the cost of both a tailored regular variant of ISD, augmented with the Stern [Ste88] and Finiasz and Sendrier [FS09] techniques, and the more recent non-regular variant due to Becker et al. [BJMM12], and then we take the minimum of the two. We have also analysed more recent variants of ISD [MO15, BM17], see Appendix B for more details.

In Table 1, we provide an estimation of the minimal key-length ℓ for our BMR-style protocols to achieve more than 128 bits of security for different values of n, h and $r = 2\ell n + 2$. Note that we only consider $8 \leq \ell \leq 32$, so when in the table we have that ℓ should be larger than 32, it means that ISD cost less than 2^{128} for that set of parameters. We also give an estimation of minimal key-length respect to the plain ISD attack to RSD by Prange.

When an RSD instance has more than one solution - this is sometimes the case for our GMW-style protocol - we need to consider also GBA and LA. Notice that since there are many solutions, attacking regular SD with classical ISD is more difficult than attacking non-regular SD and an adversary needs to run the attack repeatedly until the output is regular, increasing the cost of the attack. To estimate the complexity of GBA and LA we take the same conservative approach we use for ISD. Since LA is particularly effective for larger h, especially when $h > r/4$, we always set up $r > 2h + 1$.

In Table 2 we propose a set of of parameters for our GMW-style protocols for different values of h (and irrespective to the total number of parties n), such that the estimated complexity of the most efficient decoding algorithms is more than 2^{128}.

5.3 GMW-Style Protocol

We now compare the communication cost of our triple generation protocol with the best-known instantiation of GMW, namely a variant based on 1-out-of-4 OT to generate triples, recently optimized by [DKS+17] in the 2-party setting. This easily extends to the multi-party case with communication complexity $O(n^2\kappa/\log\kappa)$

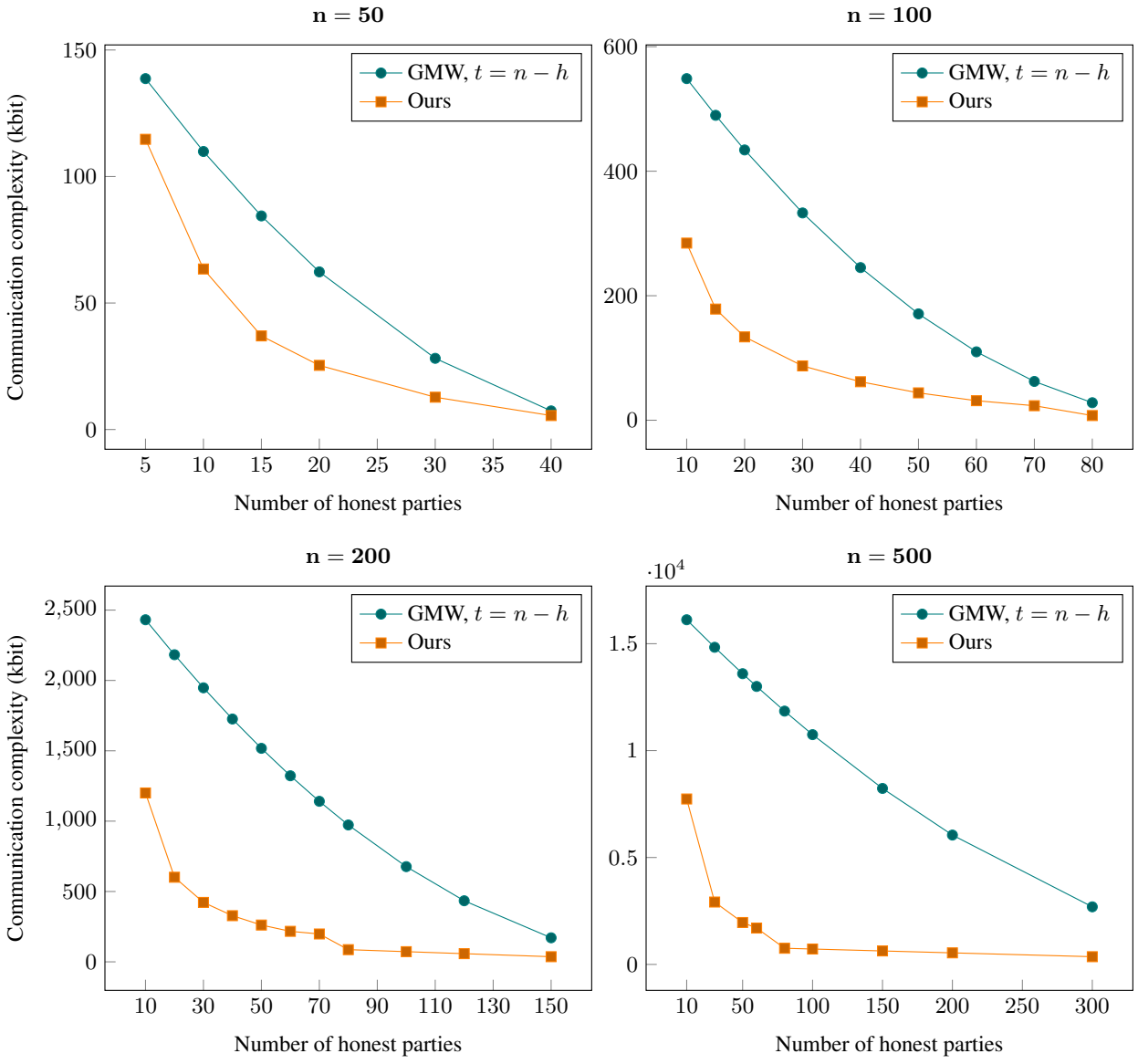

Fig. 15. Amortized communication cost (in kbit) for producing triples in GMW for $n = 50, 100, 200, 500$ and deterministic committees

bits per AND gate, so we consider both full-threshold and threshold-t (§5.1) variants. Note that our protocol from Section 3 has complexity $O(nt\ell)$ when using deterministic committees, with ℓ as in Table 1.

As can be seen in Table 3 and Figure 15, for a fixed number of honest parties h, the improvement of our protocol over GMW (threshold t) becomes greater as the total number of parties increases. Our protocol starts to beat the best-known GMW protocol for producing multiplication triples when there are just 6 honest parties. For example, with 20 parties and 14 corruptions, the communication cost of our protocol is roughly 10% lower than threshold-14 GMW, and only 2 times lower than the cost of standard, full threshold GMW. As the number of parties (and honest parties) grows, our improvements become even greater, and when the

number of honest parties is more than 80, we can use 1-bit keys and improve upon the threshold variant of GMW by *more than* 13 *times*.

In Section 3 we mentioned the possibility, when n and h are large enough, of using *random committees* $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$, such that except with negligible probability $\mathcal{P}_{(h)}$ has at least $h' \leq h$ honest parties and $\mathcal{P}_{(1)}$ has at least one honest party. In order to estimate the communication complexity of our protocol, we consider the probability $p_{(1)}$ of $\mathcal{P}_{(1)}$ not having a single honest party and the probability $p_{(h)}$ of $\mathcal{P}_{(h)}$ of having less than h' honest parties. Let $n_1 = |\mathcal{P}_{(1)}|$ and $n_h = |\mathcal{P}_{(h)}|$, we have that

$$p_{(1)} = \frac{\binom{n-h}{n_1}}{\binom{n}{n_1}} \quad \text{and} \quad p_{(h)} = \frac{\sum_{j=1}^{\min(h',h'-v)} \binom{n-h}{n_h-h'+j} \cdot \binom{h}{h'-j}}{\binom{n}{n_h}},$$

where $v = n_h - (n-h) < h'$. We fix the parameters n, h, h', and compute the minimum values n_h, n_1 such that $p_{(h)}$ and $p_{(1)}$ are less than 2^{-s}. Table 4 compares our protocol with random committees and GMW with a single random committee of size n_1, i.e. having at least one honest party with overwhelming probability, when $s = 40$. Even if the communication complexity reduces in both protocols, our approach is always at least 50% more efficient compared to GMW.

(n, h, h')	(100, 40, 30)	(200, 70, 50)	(500, 200, 120)	(800, 300, 120)	(1000, 200, 120)	(5000,1200,120)	(10000, 3000, 120)
(ℓ_{OT}, r)	(8, 300)	(6, 450)	(1, 80)	(1, 80)	(1, 80)	(1, 80)	(1, 80)
$(n_{h'}, n_1)$	(90, 39)	(180, 54)	(382, 51)	(447, 57)	(790, 117)	(811, 100)	(654, 78)
GMW	99.3	191.75	170.85	213.9	909.32	663.3	402.40
Ours	**43.6**	**84.62**	**70.13**	**91.72**	**337.75**	**291**	**183.64**

Table 4. Amortized communication cost (in kbit) of producing a single triple in GMW using random committees.

5.4 BMR-Style Protocol

Communication Complexity. To show the efficiency of our constant-round garbling protocol from Section 4.5, we provide Table 5, which has two parts. First, it compares the amortized communication complexity incurred for garbling an AND gate with [BLO16]. We recall that this is the dominating cost for BMR-style protocols using Free-XOR, and that we incur no communication for creating shares of garbled splitter gates. Note that in the first setting of $n = 20, t = 10$, although our communication costs are around 3 times lower than [BLO16], we do not improve upon the threshold-t variant of that protocol, described earlier. Once we get to 50 parties, though, we start to improve upon [BLO16], with a reduction in communication going up to 7x for 400 parties and 10x for 1000 parties.

The second half of the table shows the size of the garbled circuit in terms of the total number of AND, XOR and splitter gates. Garbled circuit size only has a slight impact on communication complexity, when opening the garbled circuit, which is much lower than the communication in the rest of the garbling phase. However, if an implementation needs to store the entire garbled circuit in memory (either for evaluation, or storage for later use) then it is also important to optimize its size. Here we also compare with [BLO17], which recently showed how to construct a compact multi-party garbled circuit based on key-homomorphic PRFs. The size of their garbled circuit is constant and grows with $O(\kappa)$ per gate, with security proven in the presence of $n-1$ corrupted parties. On the other hand, their construction requires much more expensive

# parties (honest)	20 (10)	50 (20)	80 (32)	100 (40)	200 (60)	400 (120)	1000 (160)
$(\ell_{\text{BMR}}, \ell_{\text{OT}}, r)$	$(32, 23, 530)$	$(27, 13, 450)$	$(17, 8, 380)$	$(15, 7, 400)$	$(8, 5, 370)$	$(8, 1, 80)$	$(8, 1, 120)$
[BLO16] (Gb $\mathcal{P}$)	341.24	2200.1	5675.36	8890	35740	143320.8	897102
[BLO16] (Gb $\mathcal{P}_{(1)}$)	**98.78**	835.14	2112.1	3286.7	17726.45	70654.7	634383.12
Ours (Garbling)	111.7	**747.63**	**1750.48**	**2678.74**	**5448.36**	**10114.99**	**64474.1**
[BLO16] ($\|GC\|$ $\mathcal{P}$)	$10.24A$	$25.6A$	$40.96A$	$51.2A$	$102.4A$	$204.8A$	$512A$
[BLO16] ($\|GC\|$ $\mathcal{P}_{(1)}$)	$5.632A$	$15.88A$	$25.1A$	$31.23A$	$72.19A$	$143.9A$	$430.6A$
[BLO17] ($\|GC\|$)	$12.29(A+X)$	$12.29(A+X)$	$12.29(A+X)$	$12.29(A+X)$	$12.29(A+X)$	$12.29(A+X)$	$12.29(A+X)$
Ours ($\|GC\|$)	$2.56(A+S)$	$5.4(A+S)$	$5.45(A+S)$	$6(A+S)$	$6.4(A+S)$	12.8(A+S)	32(A+S)

Table 5. Communication complexity for garbling, and size of garbled gates, in BMR-style protocols in kbit. A = #AND gates, S = #Splitter gates, X = #XOR gates.

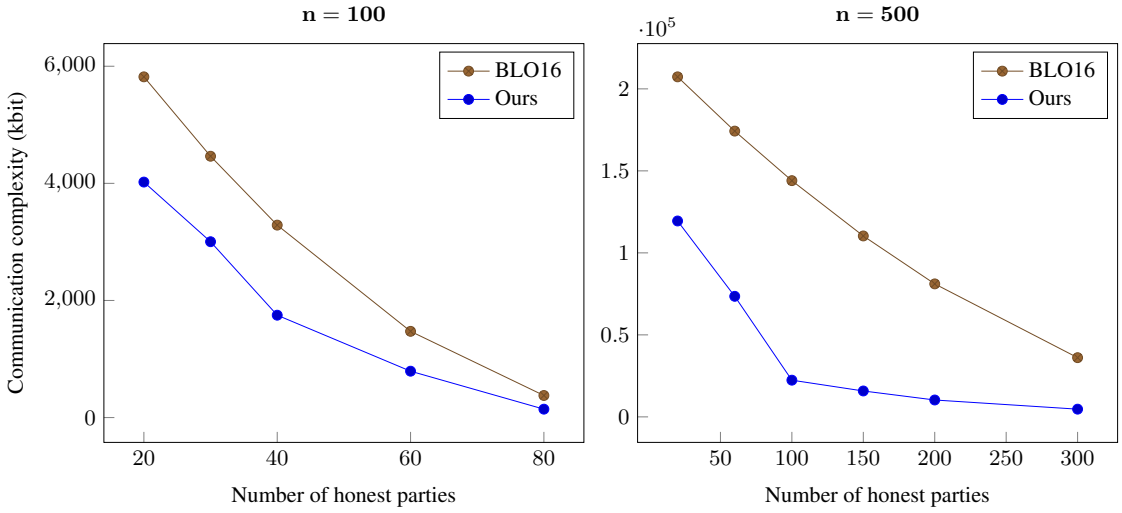

Fig. 16. Communication complexity cost (in kbit) for garbling when $n = 100$ and $n = 500$

operations based on the Decisional Diffie-Hellman (DDH) or Ring-LWE assumptions, and these also lead to fairly large keys — with a 3072-bit discrete log prime (equivalent to 128-bit security) the size of a garbled AND gate only beats our protocol at around 400 parties. Additionally, their construction does not support free-XOR, and the concrete efficiency of the offline garbling phase is not clear: garbling an AND gate requires $O(n)$ secret-shared field multiplications, which seems likely to be much higher than the offline cost of our protocol or [BLO16], but their paper does not give concrete numbers. In Figure 16 we show the communication complexity of garbling when $n = 100, 500$ and for different number of honest parties.

Garbling Implementation. In Figure 17, we present running times for evaluating the garbled circuit in our protocol and compare with times for [BLO16] running on the same machine.

The implementation runs on a single machine,[9] to allow testing just the local computation in the online phase (note that there is very little interaction in the online phase). We took as benchmarks the AES circuit (6800 AND gates, 31796 Splitters), the SHA-256 circuit (90825 AND gates, 132586 Splitters), a binary multiplier for 32-bit numbers (5926 AND gates, 6994 Splitters) and a randomly generated circuit with 100000 AND gates and 99510 Splitters (as used in [BLO17], for comparison). The CRS H was implemented with fixed-key AES in counter mode using AES-NI instructions, which is a random function under

[9] Intel Xeon E5-2650 v3 2.3GHz / 25M Cache, 10 Cores, 64GB RAM.

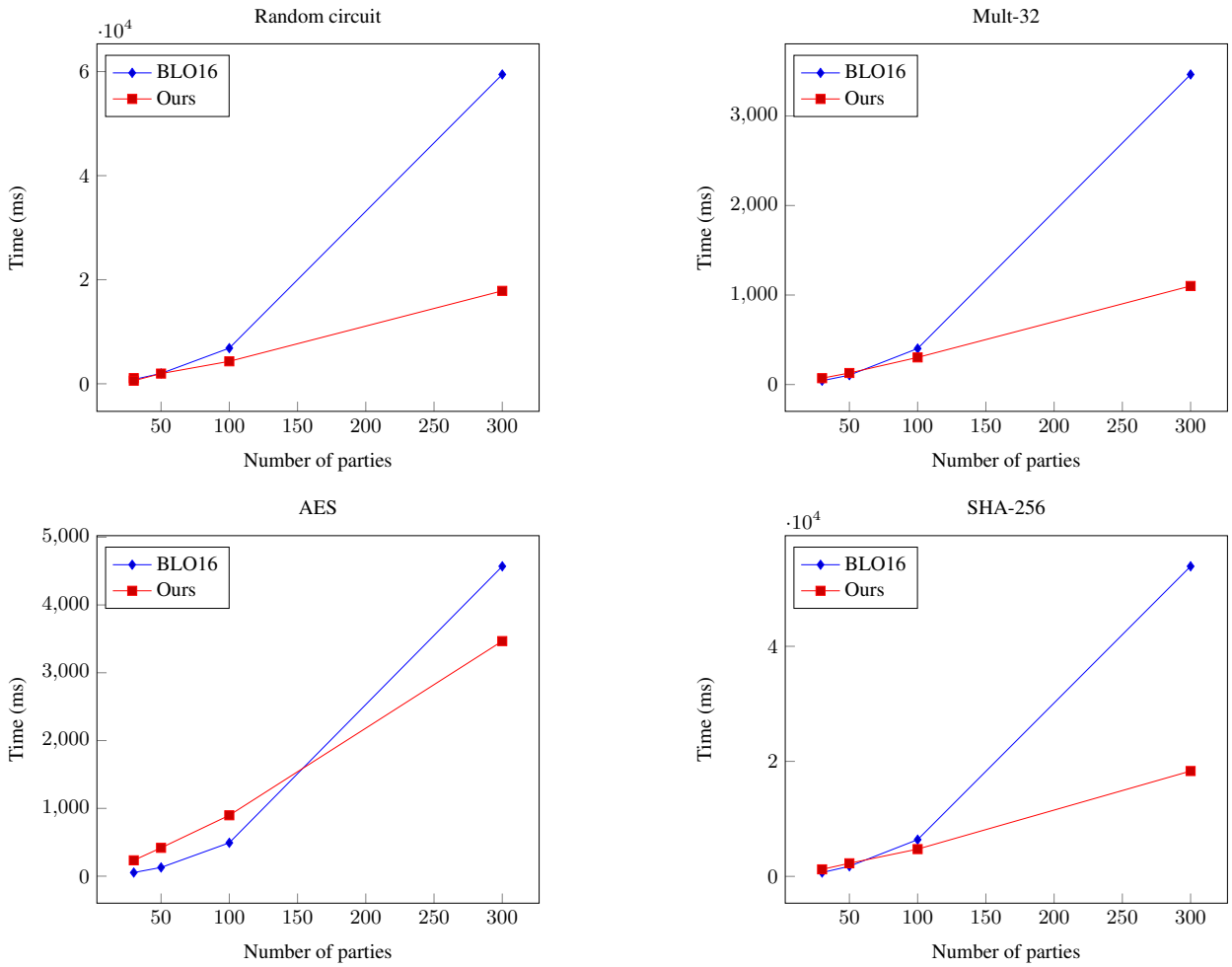

Fig. 17. Online time for evaluating various circuits with $n = 30, 50, 100, 300$. The corresponding numbers of honest parties are $h = 14, 21, 38, 105$, respectively. Times for [BLO16] are for a full-threshold implementation.

the assumption that AES behaves like a random permutation (see Appendix C). We also tried precomputing every output of H and storing this as a lookup-table, but in practice this did not perform well as the table size was usually much larger than the CPU cache. Recall that the standard BMR online phase requires each party to perform $O(n^2)$ AES operations per AND gate, whereas our online phase reduces this to $O(n^2 \ell_{\text{BMR}}/\kappa)$, with some extra cost for evaluating splitter gates. The results show that for the random circuit our protocol starts to pay off from around 50 parties, when the corruption threshold is between 20–40%, reaching a 3.3x improvement for $n = 300, h = 105$. On the other hand, for AES, which has a relatively large proportion of splitter gates, the crossover point is closer to 150 parties, and the greatest improvement factor is 1.3x over [BLO16] for $n = 300, h = 105$.

This shows that the performance improvements of our garbled circuit-based protocol very much depend on the specific circuit being evaluated, but further improvements may be possible by modifying secure computation compilers to produce circuits more suitable for our protocol. It also seems likely that implementing our GMW-based protocol would show much more significant gains, based on the communication costs presented earlier.

References

ABP11. Martin R. Albrecht, Gregory V. Bard, and Clément Pernet. Efficient dense gaussian elimination over the finite field with two elements. *CoRR*, abs/1111.6549, 2011.

ADS+17. Gilad Asharov, Daniel Demmler, Michael Schapira, Thomas Schneider, Gil Segev, Scott Shenker, and Michael Zohner. Privacy-preserving interdomain routing at internet scale. *PoPETs*, 2017(3):147, 2017.

AFL+16. Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 805–817. ACM Press, October 2016.

AFS03. Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A fast provably secure cryptographic hash function. *IACR Cryptology ePrint Archive*, 2003:230, 2003.

AIK09. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography with constant input locality. *Journal of Cryptology*, 22(4):429–469, October 2009.

AJL+12. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

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 13*, pages 535–548. ACM Press, November 2013.

App16. Benny Applebaum. Garbling XOR gates "for free" in the standard model. *J. Cryptology*, 29(3):552–576, 2016.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How 1 + 1 = 0 improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012.

BLN+09. Daniel J. Bernstein, Tanja Lange, Ruben Niederhagen, Christiane Peters, and Peter Schwabe. Fsbday. In *INDOCRYPT*, pages 18–38, 2009.

BLO16. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 578–590. ACM Press, October 2016.

BLO17. Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Efficient scalable constant-round MPC via garbled circuits. In *ASIACRYPT*, 2017.

BLP08. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. Cryptology ePrint Archive, Report 2008/318, 2008. http://eprint.iacr.org/2008/318.

BLP11. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 743–760. Springer, Heidelberg, August 2011.

BM97. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 163–192. Springer, Heidelberg, May 1997.

BM17. Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for lpn security. *IACR Cryptology ePrint Archive*, 2017:1139, 2017.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

BO17. Aner Ben-Efraim and Eran Omri. Concrete efficiency improvements for multiparty garbling with an honest majority. In *Latincrypt 2017*, 2017.

BOGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

Bra85. Gabriel Bracha. An $O(\lg n)$ expected rounds randomized byzantine generals protocol. In *17th ACM STOC*, pages 316–326. ACM Press, May 1985.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CC81. George C. Clark and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. Perseus Publishing, 1981.

CC98. Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to mceliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Trans. Information Theory*, 44(1):367–378, 1998.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

CCL15. Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

CJ04. Jean-Sébastien Coron and Antoine Joux. Cryptanalysis of a provably secure cryptographic hash function. *IACR Cryptology ePrint Archive*, 2004:13, 2004.

CS98. Anne Canteaut and Nicolas Sendrier. Cryptanalysis of the original McEliece cryptosystem. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 187–199. Springer, Heidelberg, October 1998.

DKS+17. Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *NDSS*, 2017.

DMS04. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX*, pages 303–320, 2004.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, August 2007.

Dum91. I. Dumer. On minimum distance decoding of linear codes. In *5th Joint Soviet-Swedish Int. Workshop Inform. Theory, Proceedings*, pages 50–52, 1991.

FS09. Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, Heidelberg, December 2009.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

GL89. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

Gol04. Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

HJ10. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT*, pages 235–256, 2010.

HS13. Yann Hamdaoui and Nicolas Sendrier. A non asymptotic analysis of information set decoding. *IACR Cryptology ePrint Archive*, 2013:162, 2013.

HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *ASIACRYPT*, pages 598–628, 2017.

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.

ILL89. Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st ACM STOC*, pages 12–24. ACM Press, May 1989.

Kir11. Paul Kirchner. Improved generalized birthday attack. *IACR Cryptology ePrint Archive*, 2011:377, 2011.

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.

KMR14. Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

LB88. Pil Joong Lee and Ernest F. Brickell. An observation on the security of mceliece's public-key cryptosystem. In *EUROCRYPT*, pages 275–280, 1988.

Leo88. Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Information Theory*, 34(5):1354–1359, 1988.

LPSY15. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 319–338. Springer, Heidelberg, August 2015.

LSS16. Yehuda Lindell, Nigel P. Smart, and Eduardo Soria-Vazquez. More efficient constant-round multi-party computation from BMR and SHE. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 554–581. Springer, Heidelberg, October / November 2016.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT*, pages 107–124, 2011.

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.
MS09. Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In Claire Mathieu, editor, *20th SODA*, pages 586–595. ACM-SIAM, January 2009.
NCB11. Robert Niebuhr, Pierre-Louis Cayrel, and Johannes Buchmann. Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems. In *WCC*, pages 163–172, Paris, France, 2011.
NR17. Jesper Buus Nielsen and Samuel Ranellucci. On the computational overhead of MPC with dishonest majority. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 369–395. Springer, Heidelberg, March 2017.
Pie12. Krzysztof Pietrzak. Subspace LWE. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 548–563. Springer, Heidelberg, March 2012.
Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Information Theory*, 8(5):5–9, 1962.
Saa07. Markku-Juhani Olavi Saarinen. Linearization attacks against syndrome based hashes. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 1–9. Springer, Heidelberg, December 2007.
Sen11. Nicolas Sendrier. Decoding one out of many. Cryptology ePrint Archive, Report 2011/367, 2011. http://eprint.iacr.org/2011/367.
Ste88. Jacques Stern. A method for finding codewords of small weight. In *Coding Theory and Applications*, pages 106–113, 1988.
TS16. Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In *PQCrypto*, pages 144–161, 2016.
TX03. Stephen R Tate and Ke Xu. On garbled circuits and constant round secure function evaluation. *CoPS Lab, University of North Texas, Tech. Rep*, 2:2003, 2003.
vT88. Johan van Tilburg. On the mceliece public-key cryptosystem. In *CRYPTO*, pages 119–131, 1988.
Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.
Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Universal Composability

We prove security of our protocols in the universal composability (UC) framework [Can01] (see also [CCL15] for a simplified version of UC). This framework is based on the real/ideal paradigm, where all the entities (including the parties and the adversary) are modeled as interactive Turing machines. The goal of a protocol is defined by an *ideal functionality*, which can be seen as a trusted party sending the desired results to the parties. To prove security of a protocol, we aim to show that any adversary attacking the real protocol can be used to construct a corresponding ideal adversary, called the *simulator*, that runs in the ideal world, interacting only with the functionality $\mathcal{F}$ and the real adversary, such that the distributions of messages seen in the real world and ideal world executions are indistinguishable. The UC framework additionally defines a powerful entity called the *environment*, which is the interactive machine trying to distinguish the two worlds. The environment has total control over the adversary, and can choose the inputs, and see the outputs, of *all* parties.

We denote by $\mathbf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$ the output distribution of the environment $\mathcal{Z}$ in the real world execution of protocol π, with n parties and an adversary $\mathcal{A}$, where κ is the security parameter and z is the auxiliary input to $\mathcal{Z}$. The output distribution of $\mathcal{Z}$ in the ideal world is denoted by $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)$, where $\mathcal{F}$ is the ideal functionality to be realized and $\mathcal{S}$ is the simulator. Additionally, we denote the hybrid execution of a protocol π, which is given access to an ideal functionality $\mathcal{G}$, by $\mathbf{HYB}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$. This is defined similarly to the real execution, and is known as the $\mathcal{G}$-hybrid model. Security of a protocol is then defined as follows.

Definition A.1 *A protocol π UC-securely computes an ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if for any* PPT *adversary $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{S}$ such that for any* PPT *environment $\mathcal{Z}$, it holds that:*

$$\mathbf{HYB}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

The *composition theorem* provides security guarantees when protocols are composed together. This means that if ρ is a UC-secure protocol for $\mathcal{G}$, then the protocol π in the $\mathcal{G}$-hybrid model can be replaced by the composition $\pi \circ \rho$. Informally, the composition theorem then guarantees that $\mathbf{REAL}_{\pi\circ\rho,\mathcal{A},\mathcal{Z}}$ is indistinguishable from $\mathbf{HYB}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$.

B Cryptanalysis

In this section we give a concrete analysis of the hardness of the Syndrome Decoding Problem (SD) and Regular Syndrome Decoding Problem (RSD) described in Section 2.3, in order to justify the parameters selection for our protocols. Let us start by providing useful notation on coding theory. For ease of presentation all vectors are intended to be column vectors.

A binary $[m, k, d]_2$ linear code C is a k-dimensional subspace of $\mathbb{F}_2^m$, where m is the length of the code, k is its dimension as a vector subspace and d is its distance, i.e. the minimal non-zero Hamming distance between any two codewords. Equivalently, C can be defined as the kernel of a full-rank matrix $\mathbf{H} \in \mathbb{F}_2^{(m-k)\times m}$, called a parity-check matrix of C. Given a vector $\boldsymbol{r} = \boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_2^m$, where $\boldsymbol{c} \in C$ is a codeword and $\boldsymbol{e}$ an error vector, the *syndrome* corresponding to $\boldsymbol{c}$ is the vector $\boldsymbol{s} = \mathbf{H}\cdot\boldsymbol{r} = \mathbf{H}\cdot\boldsymbol{c} + \mathbf{H}\cdot\boldsymbol{e} = \mathbf{H}\cdot\boldsymbol{e} \in \mathbb{F}_2^{m-k}$. Hence, the syndrome does not depend on the codeword, but only on the error vector. When the Hamming weight of $\boldsymbol{e}$ is smaller than the error correction capability of C, that is $\mathsf{wt}(\boldsymbol{e}) \leq \lfloor \frac{d-1}{2} \rfloor$, $\boldsymbol{s}$ is called *correctable syndrome* and $\boldsymbol{r}$ can be uniquely decoded to $\boldsymbol{c}$. More formally, we can define a mapping

$$\begin{aligned} \mathsf{Syn} : \mathbb{F}_2^m &\longrightarrow \mathbb{F}_2^r \qquad\qquad (r = m - k) \\ \boldsymbol{e} &\longmapsto \mathbf{H}\cdot\boldsymbol{e} (= \boldsymbol{s}). \end{aligned}$$

When the domain of Syn is restricted to vectors of upper bounded Hamming weight, inverting Syn is strictly related to the problem of decoding the $[m, k, d]_2$ linear code with parity-check matrix $\mathbf{H}$, and this problem is equivalent to the average-case hardness of the following computational problem.

Definition B.1 (Syndrome Decoding Problem (SD)) *Let $r, h, m \in \mathbb{N}$. Sample $\mathbf{H} \leftarrow \mathbb{F}_2^{r\times m}$ and $\boldsymbol{e} \leftarrow \mathbb{F}_2^m$ such that $\mathsf{wt}(\boldsymbol{e}) = h$, where $\mathsf{wt}(\boldsymbol{e})$ denotes the Hamming weight of $\boldsymbol{e}$. Given $(\mathbf{H}, \mathbf{H}\cdot\boldsymbol{e})$, the $\mathsf{SD}_{r,m,h}$ problem is to recover $\boldsymbol{e}$ with noticeable probability.*

Notice SD can be seen as a purely combinatorial problem and the most näive algorithm could simply enumerate all the $\binom{m}{h}$ possible solutions.

We denote by $W_{m,h}$ the set of all the binary vectors in $\mathbb{F}_2^m$ of weight exactly h and by $R_{m,h} \subset W_{m,h}$ the set of all the binary (m, h)-regular vectors. We recall that a binary (m, h)-regular vector is a vector in $\mathbb{F}_2^m$ such that, if we divide it into h blocks of equal length, each of them has Hamming weight exactly 1. When SD is restricted to vectors in $R_{m,h}$, we have the following.

Definition B.2 (Regular Syndrome Decoding (RSD)) *Let $r, h, \ell \in \mathbb{N}$ and $m = h \cdot 2^\ell$. Sample $\mathbf{H} \leftarrow \mathbb{F}_2^{r\times m}$ and $\boldsymbol{e} \leftarrow R_{m,h}$. Given $(\mathbf{H}, \mathbf{H}\cdot\boldsymbol{e})$, the $\mathsf{RSD}_{r,h,\ell}$ problem is to recover $\boldsymbol{e}$ with noticeable probability.*

This problem was introduced in 2003 by Augot et al. [AFS03], who used it for the SHA-3 candidate FSB (Fast Syndrome-Based) hash function. As for SD, RSD can be seen as a combinatorial problem and solved by enumerating all the $(m/h)^h$ possible solutions. The most efficient attacks against SD and RSD

are Information Set Decoding (ISD), Generalised Birthday Attack (GBA) and Linearization Attack (LA). To establish which attack is the most efficient we need to distinguish different cases depending on the choice of h. Clearly, the number of errors affect the practical security of SD and RSD. In coding theory, the values of h usually considered are those corresponding to a single solution of the problem, which happens with high probability if the number of errors h is smaller than the Gilbert-Varshamov distance d_{GV}.[10] In cryptography, there is no restriction on h as long as SD or RSD remain hard. In practice, when $h \leq d_{GV}$, ISD is always the most efficient attack and has roughly the same cost when considering SD and RSD; when $h > d_{GV}$ the best attack is either ISD or GBA. Furthermore, when considering the regular case, the intuition is that if the solution to the problem is not unique, attacking RSD is even harder than attacking SD: having a smaller set of inputs decreases the number of possible solutions and thus increases the cost of the attacks. Finally, when h is larger, say $h > r/4$, the best attack is linearization.

In the rest of this section we analyse all the most efficient attacks to SD and RSD.

B.1 Linearization Attack

This attack was described in [Saa07] and is a generalisation of [BM97, Appendix A]. It is very efficient for h large, namely when $h > r/4$, and consists in finding linear relations between h columns of $\mathbf{H}$.

Algorithm 1 Linearization Attack

Input: $\mathbf{H} \leftarrow \mathbb{F}_2^{r \times m}$, $\boldsymbol{s} \in \mathbb{F}_2^r, h \in \mathbb{N}$
Output: $\boldsymbol{e} \in \mathbb{F}_2^m$ such that $\mathsf{wt}(\boldsymbol{e}) = h$ and $\mathbf{H}\boldsymbol{e} = \boldsymbol{s}$
Parameters: $1 \leq \lambda \leq r/h$

1: **repeat**
2: Choose h columns, $\boldsymbol{h}^{a_1^0}, \ldots, \boldsymbol{h}^{a_h^0}$ of $\mathbf{H}$, one for each block, i.e. $a_i^0 = (i-1) \cdot 2^\ell + t_i^0$, $i \in [h]$ and $1 \leq t_i^0 \leq 2^\ell$, and compute $\sum_i \boldsymbol{h}^{a_i^0} + \boldsymbol{s} = \Delta$
3: For $j \in [\lambda]$ choose $t_j^1, \ldots, t_j^h$, such that $1 \leq t_j^i \leq 2^\ell$
4: $\forall (i,j) \in [h] \times [\lambda]$, compute the differences $\boldsymbol{\delta}_i^j = \boldsymbol{h}^{a_j^0} + \boldsymbol{h}^{a_j^i}$, where $\boldsymbol{h}^{a_i^j} = \boldsymbol{h}^{(i-1)2^\ell + t_i^j}$
5: **until** $\exists\, \epsilon_i^j \in \{0,1\}, (i,j) \in [h] \times [\lambda]$, such that $\sum_{i=1}^h \sum_{j=1}^\lambda \epsilon_i^j \cdot \boldsymbol{\delta}_i^j = \Delta$ and for each i there is at most one $\epsilon_i^j = 1$ among the λ of them.

A description of this attack is given in Algorithm 1. As we are considering the RSD problem, we can assume that the matrix $\mathbf{H}$ consists of h blocks of 2^ℓ columns, i.e. $\mathbf{H} = [\mathbf{H}_1 \mid \cdots \mid \mathbf{H}_h]$. The adversary chooses a parameter $\lambda \leq r/h$ and h columns of $\mathbf{H}$, say $\boldsymbol{h}^{a_1^0}, \ldots, \boldsymbol{h}^{a_h^0}$, in such a way that only one column for each of the h blocks is chosen. Then they compute $\Delta = \sum_i \boldsymbol{h}^{a_i^0} + \boldsymbol{s}$ and, for each $j \in [\lambda]$, choose a different set of h columns, $\boldsymbol{h}^{a_1^j}, \ldots, \boldsymbol{h}^{a_h^j}$, of $\mathbf{H}$. After that, using linear algebra, they looks for linear relations $\sum_{i=1}^h \sum_{j=1}^\lambda \epsilon_i^j \cdot \boldsymbol{\delta}_i^j = \Delta$, with $\epsilon_i^i \in \{0,1\}$. As $\lambda \leq r/w$, all the possible linear combinations of the elements in $\{\boldsymbol{\delta}_i^j\}_{(i,j) \in [h] \times [\lambda]}$ form a vector space of dimension $h \cdot \lambda \leq r$ at most. Hence, the desired linear relation exists with probability $2^{h\lambda}/2^r$. A relation is useful only with probability $((\lambda+1)/2^\lambda)^h$, as for each $i \in [h]$ it should involve at most one of the $\boldsymbol{\delta}_i^j$ values. More concretely and for a fixed i, as each ϵ_i^j is set i.i.d. with probability $1/2$, only $\lambda + 1$ cases satisfy the restriction on the $\boldsymbol{\delta}_i^j$ values, each of those cases having probability $1/2^\lambda$. Overall, this means that the expected number of iterations to find a useful relation is slightly above $\frac{2^r}{(\lambda+1)^h}$. When $h \leq r/2\lambda$, it is possible to consider 2λ generators, so that, using the same

[10] The Gilbert-Varshamov (GV) distance is the largest integer d_{GV}, such that $\sum_{i=0}^{d_{GV}-1} \binom{m}{i} \leq 2^r$.

arguments as above, the probability of a useful relation is $(\lambda+1)^{2h}/2^r$. Summing up:

$$\mathsf{C}_{\mathsf{LA}} \geq \begin{cases} \frac{2^r}{(\lambda+1)^h} & \text{if} \quad h \leq r/\lambda \\ \frac{2^r}{(\lambda+1)^{2h}} & \text{if} \quad h \leq r/2\lambda \end{cases}$$

B.2 Generalised Birthday Attack

Algorithm 2 Wagner's GBA

Input: $t = 2^a$ lists $L_0, \ldots, L_{t-1}$ containing uniform random elements from $\mathbb{F}_2^r$

Output: $\boldsymbol{h}_i \in L_i, \forall i \in [t]$ such that $\sum_i \boldsymbol{h}_i = 0$

Let $L_{0,0}, \ldots, L_{0,2^a-1}$ be 2^a lists of elements in $\mathbb{F}_2^r$, where $L_{i,j}$ denotes the jth list on level ith, with $S = |L_{ij}| = 2^{\frac{r}{a+1}}$.

Level 0. $L_{1,j} \leftarrow \mathsf{Merge}(L_{0,2j}, L_{0,2j+1})$: Compute $L_{0,2j} + L_{0,2j+1}$ and consider only sums of two vectors starting with $\frac{r}{a+1}$ zeros. Result of this step are 2^{a-1} lists $L_{1,j}$.

Level 1. $L_{2,j} \leftarrow \mathsf{Merge}(L_{1,2j}, L_{1,2j+1})$: Sum elements starting with $2\frac{r}{a+1}$ zeros. The result of this step are 2^{a-2} lists $L_{2,j}$.

Level i. Pairwise merge lists from level $i-1$ by considering elements starting with $(i+1)\frac{r}{a+1}$ zeros.

Level a − 2. Proceed as before. As a result we obtain 2 lists containing $2^{\frac{r}{a+1}}$ elements whose first $(a-1)\frac{r}{a+1}$ components are zeros.

Output: Only $2\frac{r}{a+1}$ components in the two remaining lists are non-zero, apply the standard birthday technique to find a solution.

This attack is named after the famous birthday paradox which permits to find collisions between two random lists much faster than checking every possible combinations. GBA improves the standard birthday paradox by looking for specific solutions and discarding the others, so it does not apply when a single solution to SD/RSD exists.

Formally, the *Generalised Birthday Problem* (GBP) can be stated as follows: given t lists $L_1, \ldots, L_t$, containing uniform random elements from $\mathbb{F}_2^r$, find exactly one element $\boldsymbol{h}^i$ in each list L_i, $i \in [t]$, such that $\sum_{i \in [t]} \boldsymbol{h}^i = 0$.

Note that in this definition the number of available vectors is unbounded, but when one considers $\mathsf{SD}_{r,m,h}$ or $\mathsf{RSD}_{r,h,\ell}$, only m different r-bit strings are available, that is the m columns of $\mathbf{H}$. Also, when the syndrome $\boldsymbol{s} \neq 0$, we can simply subtract $\boldsymbol{s}$ from each element of one list, say L_1, and proceed to find a combination of elements that sum up to zero as in GBP.

To solve GBP, Wagner proposed the so called k-tree algorithm. It consists in a divide and conquer approach: at each step it uses only two lists at time, performing a simple collision search. This procedure requires a large number of inputs and therefore more memory than the classic birthday algorithm. Minder et al. [MS09] extended the k-tree algorithm offering a solution which allows to balance time and memory efficiency of the attack. This technique was firstly applied to decoding problems by Coron and Joux [CJ04] and then improved by [BLN+09, Kir11, NCB11].

For ease of exposition we only consider the case $t = 2^a$, however the attack still works if this is not the case, only with a slight loss of efficiency. In Algorithm 2, we give a more detailed description of the algorithm, as described by Wagner. The algorithm starts by building 2^a lists of size $S = 2^{\frac{r}{a+1}}$ and consists of a steps. Each element in a list is a vector in $\mathbb{F}_2^r$ and can be seen as a concatenation of $a+1$ elements in $\mathbb{F}_2^{\frac{r}{a+1}}$, so that each element $\boldsymbol{h}$ in a list can be visualised as

$$(h_1, \ldots, h_{\frac{r}{a+1}} \| h_{\frac{r}{a+1}+1}, \ldots, h_{\frac{2r}{a+1}} \| h_{\frac{ar}{r+1}+1}, \ldots, h_r).$$

At each step elements in these lists are pairwise merged so that the number of lists is halved. More precisely, at level 0, we start with $t = 2^a$ lists $L_{0,0}, \ldots, L_{0,2^a-1}$ and from these we obtain $L_{1,0}, \ldots, L_{1,2^a-2}$ by computing $L_{1,j} = L_{0,2j} + L_{0,2j+1}$ and considering only sums of vectors with the first block of $\frac{r}{a+1}$ coordinates equal to zero. Clearly, with this condition some possible solutions will be discarded, but it permits to maintain the size of the lists equal to S on average. Moreover, those zero coordinates are not considered in the next level. This operation is repeated until only 2 lists remain and a standard birthday technique can be applied to find a solution. Since this solution is the sum of 2^a elements from the original lists, it is also a solution for the GBP.

A first observation is that the last step, i.e. the standard birthday technique, will be successful as long as the size S of the lists satisfies $S^2 \geq 2^{r-(a-1)\log_2 S}$, that is $S \geq 2^{\frac{r}{a+1}}$. Therefore, if it is possible to build lists of size $S = 2^{\frac{r}{a+1}}$, then the complexity of the algorithm can be lower bounded by $O(S \cdot \log_2 S)$. Now, if we use GBA to solve $\mathsf{SD}_{r,m,h}$ or $\mathsf{RSD}_{r,h,\ell}$, the size of the starting lists is bounded by the number of columns m of $\mathbf{H}$. As said above, the size S of these lists needs to verify $S \geq 2^{\frac{r}{a+1}}$, hence, assuming that the starting lists contain sums of $\frac{h}{2^a}$ columns of $\mathbf{H}$, we need:

$$\binom{m}{\frac{h}{2^a}} \geq 2^{\frac{r}{a+1}}.$$

Moreover, we need the lists to contain different values, so it should be

$$\binom{S}{2^a} \leq \binom{m}{h},$$

from which we obtain:

$$\frac{2^a}{a+1} \leq \frac{\log_2(\binom{m}{h})}{r}. \tag{3}$$

When the solutions are regular, each list is associated with $h/2^a$ blocks, each of weight 1, of $\mathbf{H}$ and S cannot be larger than the number of words of weight $h/2^a$. Since in each block there are m/h words of weight 1, then we need

$$S \leq \left(\frac{m}{h}\right)^{h/2^a}$$

and (3) becomes

$$\frac{2^a}{a+1} \leq \frac{h}{r}\log_2\left(\frac{m}{h}\right). \tag{4}$$

When we consider security against GBA, we also take into account improvements due to Niebuhr et al. [NCB11], Bernstein et al. [BLN+09] and Kirchner [Kir11].

B.3 Information Set Decoding Attacks

Information Set Decoding (ISD) can be seen as a class of generic algorithms for solving SD for random linear codes. Here we assume an SD (respectively an RSD) instance with parameters (r, m, h), associated with the $[m, k = m - r]_2$ binary linear code with parity-check matrix $\mathbf{H}$. This decoding technique was introduced by Prange in 1962 [Pra62] and later improved, by a polynomial factor, by Lee and Bricknell [LB88] and Leon [Leo88]. In 1988, Stern [Ste88] proposed an exponential improvement, followed by further improvements by Dumer [Dum91], Finiasz and Sendrier [FS09] and Bernstein et al. [BLP11]. In this section we describe a common framework for different variants of ISD and analyse their complexity. We consider the Stern-Dumer variant, with the optimisation due to Finiasz and Sendrier [FS09], and also we take into account

more recent variants, more precisely, those described by May et al. [MMT11], Becker et al. [BJMM12] and May et al. [MO15].

The high level idea of these algorithms is to transform the original SD instance with parameters (r, m, h) into a related instance with smaller parameters (r', m', h') and then try to find a solution of the original problem from a solution of the easier instance. ISD essentially applies linear algebra transformations to the parity-check matrix $\mathbf{H}$ in order to obtain a new structured matrix $\mathbf{H}'$, and reduce the space of solutions of the original SD instance. The overall idea of ISD is given in Algorithm 3.

Algorithm 3 ISD Framework

Input: $\mathbf{H} \leftarrow \mathbb{F}_2^{r\times m}$, $\boldsymbol{s} \in \mathbb{F}_2^r, h \in \mathbb{N}$
Output: $\boldsymbol{e} \in \mathbb{F}_2^m$ such that $\mathsf{wt}(\boldsymbol{e}) = h$ and $\mathbf{H}\boldsymbol{e} = \boldsymbol{s}$
Parameters: $0 \le q \le r, \quad 0 \le p \le k+q$

1: **repeat**
2: $\quad \mathbf{H}' \leftarrow \mathsf{G}(\mathbf{H})$: Pick $\mathbf{P} \leftarrow \mathbb{F}_2^{m\times m}$ and compute $\mathbf{H}' = \mathbf{UHP} = \left[\begin{array}{c|c} \mathbf{R}_1 & \mathbf{0}_q \\ \hline \mathbf{R}_2 & I_{r-q} \end{array}\right]$,
$\quad$ where $\mathbf{R}_2 \in \mathbb{F}_2^{(r-q)\times(k+q)}, \mathbf{R}_1 \in \mathbb{F}_2^{q\times(k+q)}$ and $\boldsymbol{s}' = \mathbf{U}\boldsymbol{s} = \left[\begin{array}{c} \boldsymbol{s}_1 \\ \hline \boldsymbol{s}_2 \end{array}\right]$,
$\quad$ with $\mathbf{U} \in \mathbb{F}_2^{r\times r}$ invertible, $\boldsymbol{s}_2 \in \mathbb{F}_2^{r-q}$ and $\boldsymbol{s}_1 \in \mathbb{F}_2^q$.
3: $\quad E \leftarrow \mathsf{SM}(\mathbf{R}_1, \boldsymbol{s}_1, p)$: Compute the set $E \subset \{\boldsymbol{e}_1 \in \mathbb{F}_2^{k+q} \mid \mathsf{wt}(\boldsymbol{e}_1) = p, \mathbf{R}_1 \cdot \boldsymbol{e}_1 = \boldsymbol{s}_1\}$
4: $\quad$ **for all** $\boldsymbol{e}_1 \in E$ **do**
5: $\quad\quad \boldsymbol{e}_2 \leftarrow \mathsf{Extend}(\mathbf{R}_2, \boldsymbol{e}_1, \boldsymbol{s}_2)$: Compute $\boldsymbol{e}_2 = \mathbf{R}_2 \cdot \boldsymbol{e}_1 + \boldsymbol{s}_2$
6: **until** $\mathsf{wt}(\boldsymbol{e}_2) = h - p$

The algorithm consists of three steps:

1. First pick a random $m \times m$ permutation matrix $\mathbf{P}$ (this is equivalent to choosing a uniform random subset of k columns of $\mathbf{H}$) and then performing a Gaussian elimination to get a structured matrix

$$\mathbf{H}' = \left[\begin{array}{c|c} \mathbf{R} & \begin{array}{c}\mathbf{0}_q \\ \\ \mathbf{I}_{r-q}\end{array} \end{array}\right] = \left[\begin{array}{c|c} \mathbf{R}_1 & \mathbf{0}_q \\ \hline \mathbf{R}_2 & \mathbf{I}_{r-q} \end{array}\right], \quad \mathbf{R} \in \mathbb{F}_2^{r\times(k+q)}.$$

   $\mathbf{H}'$ consists of two blocks of columns $\mathcal{R} = [1, k+q]$ and $\mathcal{I} = [k+q+1, m]$. Essentially, this step is equivalent to finding an invertible matrix $\mathbf{U} \in \mathbb{F}_2^{r\times r}$ such that $\mathbf{H}' = \mathbf{UHP}$, and forces a $q \times (r-q)$ zero block in the first q rows in the second block of columns $\mathcal{I}$. Notice the multiplication by $\mathbf{P}$ permutes the columns of $\mathbf{H}$ and hence the coordinates of $\boldsymbol{e}$ so that $\boldsymbol{e}' = P\cdot\boldsymbol{e}$, and the multiplication by $\mathbf{U}$ permutes the coordinates of $\boldsymbol{s}$, so that we get $\boldsymbol{s}' = \mathbf{U}\boldsymbol{s} = (\boldsymbol{s}_1, \boldsymbol{s}_2)$. We denote $\boldsymbol{e}' = (\boldsymbol{e}_1, \boldsymbol{e}_2)$, $\boldsymbol{e}_1 \in \mathbb{F}_2^{k+q}, \boldsymbol{e}_2 \in \mathbb{F}_2^{r-q}$.
2. Next, fix a weight $0 \le p \le h$, and consider the first slice $[\mathbf{R}_1|\mathbf{0}_q]$ of $\mathbf{H}'$, containing the first q rows. Compute SM, by creating a set of partial solutions E, that is a set of vectors $\boldsymbol{e}_1 \in \mathbb{F}_2^{k\times q}$ s.t. $\mathsf{wt}(\boldsymbol{e}_1) = p$ and $\mathbf{R}_1 \cdot \boldsymbol{e}_1 = \boldsymbol{s}_1$.
3. Extend partial solutions obtained in the previous step by computing for each $\boldsymbol{e}_1 \in E$ the vector $\boldsymbol{e}_2 = \mathbf{R}_2\boldsymbol{e}_1 + \boldsymbol{s}_2$. If $\mathsf{wt}(\boldsymbol{e}_2) = h - p$, then return $\boldsymbol{e}' = (\boldsymbol{e}_1, \boldsymbol{e}_2)$ and stop.

Note that this algorithm always outputs a correct solution:

$$\mathbf{H}'\boldsymbol{e}' = \left[\begin{array}{c|c} \mathbf{R}_1 & \mathbf{0}_q \\ \hline \mathbf{R}_2 & \mathbf{I}_{r-q} \end{array}\right] \begin{bmatrix} \boldsymbol{e}_1 \\ \\ \boldsymbol{e}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1\boldsymbol{e}_1 \\ \mathbf{R}_2\boldsymbol{e}_1 + \boldsymbol{e}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_1 \\ \\ \boldsymbol{s}_2 \end{bmatrix} = \boldsymbol{s}',$$

where $\mathbf{H}'\boldsymbol{e}' = \mathbf{UHP}\boldsymbol{e}' = \mathbf{UH}\boldsymbol{e} = \boldsymbol{s}' \Leftrightarrow \mathbf{H}\boldsymbol{e} = \mathbf{U}^{-1}\boldsymbol{s}' = \boldsymbol{s}$ and $\mathsf{wt}(\boldsymbol{e}) = \mathsf{wt}(\mathbf{P}\boldsymbol{e}') = h$. ISD contains two parameters: p, with $0 \leq p \leq h$, representing the number of errors in the selected $k+q$ columns, and q, $0 \leq q \leq r$, representing the coordinates where the error components are zero. The case $p = q = 0$ corresponds to the plain information-set decoding algorithm due to Prange.

The main difference among different ISD variants, and also the main computational task of the algorithm, is the second step, where SM is performed. Here the main problem is to find p columns of $\mathbf{R}_1$ that sum to $\boldsymbol{s}_1$. More in particular, let $\mathbf{R}_1 = [\boldsymbol{r}_1^1, \ldots, \boldsymbol{r}_1^{k+q}]$, $\boldsymbol{r}_1^j \in \mathbb{F}_2^q$, we have to find an index set $\mathcal{J} \subseteq \mathcal{R}$, with $|\mathcal{J}| = p$ and $\sum_{j\in\mathcal{J}} \boldsymbol{r}_1^j = \boldsymbol{s}_1$. This problem was called by May et al. [MMT11] the *submatrix matching problem* (SM).

We are going to examine the cost of ISD depending on how the SM step is instantiated, considering both regular and non-regular syndrome decoding.

Complexity of ISD We essentially follow and simplify the analysis done in [FS09, Sen11, HS13, TS16], but we also consider in more details the regular case. We stress that our goal is to find a reasonable estimation of the complexity of recent ISD algorithms and that a detailed analysis of these algorithms and their cost is out of the scope of this paper.

To lower bound the cost, $\mathsf{C}_{\mathsf{ISD}}$, of the attack, we need to estimate the number, NI, of iterations needed before the algorithm successfully stops. If we assume that those iterations are independent, then NI is simply the reciprocal of the success probability P of one iteration. In particular, when $h \leq d_{GV}$, that is when the solution of SD/RSD is unique, $\mathsf{NI}^{\mathsf{single}} = 1/\mathsf{P}$; when $h > d_{GV}$, $\mathsf{NI}^{\mathsf{mult}} = 1/(\mathsf{P} \cdot \mathsf{N\_Sol})$, where $\mathsf{N\_Sol}$ is the expected number of solutions. We have that $\mathsf{N\_Sol} = \frac{\binom{m}{h}}{2^r}$ in the non-regular case and $\mathsf{N\_Sol}^R = \frac{(m/h)^h}{2^r}$ for RSD.

Hence, an attack against SD will succeed if one finds an error $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ such that $\boldsymbol{e}_1 = p$ and $\boldsymbol{e}_2 = h - p$, which happens with probability

$$\mathsf{P}^{\mathsf{single}} = \frac{\binom{k+q}{p} \cdot \binom{r-q}{h-p}}{\binom{m}{h}} \quad \text{and} \quad \mathsf{P}^{\mathsf{mult}} = \frac{\binom{k+q}{p} \cdot \binom{r-q}{h-p}}{2^r},$$

where $\mathsf{P}^{\mathsf{single}}$ and $\mathsf{P}^{\mathsf{mult}}$ denote the success probability of one iteration when $h \leq d_{GV}$ and $h > d_{GV}$, respectively.

Each iteration of ISD chooses $r - q$ columns of $\mathbf{H}$ and applies Gaussian elimination to obtain I_{r-q}. The cost of Gaussian elimination is essentially the same for all the different ISD variants. This cost can be reduced using for example the *bit-swapping technique* introduced by Omura (see [CC81]) and used in ISD by vanTilburg [vT88], Canteaut and Chabaud [CC98] and Canteaut and Sendrier [CS98]. A generalisation of bit-swapping is introduced by Bernstein et al. [BLP08] which improves the balance between the cost of Gaussian Elimination and the cost of error-searching. The idea is that of starting a new iteration with the matrix $\mathbf{H}'$ from previous iteration and then, instead of choosing a new set of $r - q$ columns, reusing $r - q - t$ out of $r - q$ columns from previous iteration and selecting only t new columns randomly. In this way only t columns need to be pivoted. However, as observed by Bernestein et al., small values of t

introduce a dependence between iterations and require more iterations before the algorithm succeeds. Analysing the impact of t in the algorithm is very difficult as the iterations are no more independent and the number of errors in the selected r columns is correlated with the number of errors in the columns selected in the next iteration. So, to estimate the cost of the algorithm, we should consider a Markov chain for the number of errors, as in [CC98]. For this reason, we assume the iterations to be independent and that this step approximately costs $(r-q)^3/\log_2(r-q)$ [ABP11]. Also, we would like to remark that the cost of Gaussian elimination is most of the time negligible compared to other operations.

Next, we need to consider the number of expected iterations of Extend. Following the same arguments as in [FS09, TS16], we have that the probability of success is

$$\mathsf{P}^{\mathsf{single}}_{\mathsf{Success}} \approx \frac{2^q \cdot \binom{r-q}{h-p}}{\binom{m}{w}} \quad \text{and} \quad \mathsf{P}^{\mathsf{mult}}_{\mathsf{Success}} \approx \frac{2^q \cdot \binom{r-q}{h-p}}{2^r} \,. \tag{5}$$

Putting everything together, the average cost of ISD is

$$\mathsf{C}_{\mathsf{ISD}} \approx \min_{p,q}\Big\{\frac{\min\{2^r, \binom{m}{h}\}}{\binom{r-q}{h-p}} \cdot \Big(\frac{K_{\mathsf{Gauss}} + K_{\mathsf{SM}}}{\binom{k+q}{p}} + \frac{K_{\mathsf{Ext}}}{2^q}\Big)\Big\},$$

where K_{SM} and K_{Ext} denote the average cost in elementary binary operations of computing SM and Extend, respectively, and both depend on the ISD variant. The impact of K_{Ext} on the cost of the attack is relative small, it is essentially the cost of testing $\boldsymbol{e}_2 = \mathbf{R}_2 \cdot \boldsymbol{e}_1 + \boldsymbol{s}_2$ for $\boldsymbol{e}_1 \in E$. If we consider the plain information set decoding described by Prange, we have $p = q = 0$ and obtain:

$$C^{\mathsf{single}}_{\mathsf{ISD\_Pra}} > \frac{\binom{m}{h}}{\binom{r}{h}} \qquad C^{\mathsf{mult}}_{\mathsf{ISD\_Pra}} > \frac{2^r}{\binom{r}{h}} \tag{6}$$

In Stern and Dumer's algorithm, with Finiasz and Sendrier optimization, the submatrix matching problem is solved using a birthday collision search. More precisely, the first block of $k+q$ coordinates $\mathcal{R}$ is split in two disjoint sets $\mathcal{R}_1 = [1, \frac{k+q}{2}]$ and $\mathcal{R}_2 = [\frac{k+q}{2} + 1, k+q]$ and then by using a birthday technique, one looks for two sets $I_1 \subset \mathcal{R}_1$ and $I_2 \subset \mathcal{R}_2$ of cardinality $p/2$ such that $\sum_{i \in I_1} \boldsymbol{r}_2^i = \sum_{i \in I_2} \boldsymbol{r}_2^i + \boldsymbol{s}_1$. Roughly, the initial lists have size $\binom{(k+q)/2}{p/2}$, so $K^{\mathsf{SD}}_{\mathsf{SM}} > \binom{(k+q)/2}{p/2}$. In 2012 May et al. [MMT11] exponentially improved ISD by using the *representation technique* introduced by Howgrave-Graham and Joux [HJ10] in the context of subset sum algorithms. We don't give further details of this ISD variant, but it uses four initial lists of size $\binom{(k+q)/2}{p/4}$, so $K^{\mathsf{ISD\_MMT}}_{\mathsf{SM}} > \binom{(k+q)/4}{p/4}$. In a further work of May et al. [BJMM12], the birthday decoding is replaced by an order 3 generalised birthday decoding with 8 initial lists of size larger that $\binom{(k+q)/2}{p/8}$. The algorithm described in a more recent work by May et al. [MO15] it is slightly different from previous variants. It has the same main loop with the Gaussian Elimination, then it starts the BJMM variants with 8 lists, but in the tree structure the last join is replaced by a "Nearest Neighbours" (NN) search. This technique provides a significant asymptotic advance, but the analysis hides a large factor and it does not lead to a significant practical improvement. Finally, we would like to mention a very recent work by Both and May [BM17], which uses the same approach of [BJMM12] and the power of NN search in each step of the algorithm, and not only in the last join like [MO15]. However, the practical impact of this new approach is still unclear and it leads to a significant asymptotic improvement compared to previous attacks only when the error rate is high, which is never the case in our protocol.

In practice to analyse the complexity of ISD, we use a lower bound on BJMM, simplyfing the analysis done in [HS13]. Notice that in our construction we are mostly interested in solving $\mathsf{RSD}_{r,h,\ell}$ with $m = 2^\ell \cdot h$.

To estimate the cost of attacking RSD with ISD, we first consider the case $p = q = 0$. In this case an attacker can choose a different strategy: here the error $\boldsymbol{e}$ is regular, so it can be decomposed in h blocks with m/h components and only one error for each block, hence to maximise the probability of success when choosing the first k coordinates, the adversary should choose the same number of positions k/h from each block. This means that they take k/h coordinates out of m/h. Given that only 1 of these is different from 0, the probability of ending up with an error $\boldsymbol{e}' = (\boldsymbol{e}_1, \boldsymbol{e}_2)$ with $\mathsf{wt}(\boldsymbol{e}_1) = 0$ and $\mathsf{wt}(\boldsymbol{e}_2) = h$ is:

$$\begin{aligned}\mathsf{P}^{0,0,\mathsf{single}} &= \Pr(\{\boldsymbol{e} = (\boldsymbol{e}_1, \boldsymbol{e}_2) \in R_{m,h} \mid \mathsf{wt}(\boldsymbol{e}_1) = 0, \mathsf{wt}(\boldsymbol{e}_2) = h\}) \\ &= \left(\frac{\binom{(m/h)-1}{k/h}}{\binom{m/h}{k/h}}\right)^h = \left(\frac{r}{m}\right)^h,\end{aligned}$$

which correspond to $\mathsf{C}^{R,\mathsf{single}}_{\mathsf{ISD\_Pra}}$, i.e. the cost of Prange's algorithm for the regular case, and similarly, when the number of solutions is greater than one,

$$\mathsf{P}^{0,0,\mathsf{mult}} = \left(\frac{r}{h}\right)^h \cdot 2^{-r},$$

which, as before, correspond to $\mathsf{C}^{R,\mathsf{mult}}_{\mathsf{ISD\_Pra}}$ for the regular case. Note that this probability is smaller than for the SD, so finding a solution for RSD is harder. This strategy can be easily generalised to the case p, q different from zero. To solve RSD, the adversary could select the first block of $k + p$ coordinates as follows: take p blocks, i.e. $p \cdot m/h$ coordinates, and then fill the remaining $(k + q) - p \cdot m/h$ coordinates by following the same strategy as before, i.e. take the same number of positions a from each remaining block, where

$$a = \frac{(k + q) - p \cdot m/h}{h - p}.$$

Notice we are assuming $k + q \leq p \cdot m/h$ and a integers. There are $h - p$ remaining blocks, each with m/h components of which only one is non-zero, therefore the above probability becomes

$$\mathsf{P}^{\mathsf{R},\mathsf{single}} = \binom{h}{p} \cdot \left(\frac{\binom{(m/h)-1}{a}}{\binom{m/h}{a}}\right)^{h-p} = \binom{h}{p} \cdot \left(\frac{h \cdot (r - q)}{m \cdot (h - p)}\right)^{h-p}.$$

and

$$\mathsf{P}^{\mathsf{R},\mathsf{mult}} = \binom{h}{p} \cdot \left(\frac{r - q}{h - p}\right)^{h-p} \cdot \left(\frac{m}{h}\right)^p \cdot 2^{-r}.$$

Notice these are exactly $\mathsf{P}^{0,0}$ when $p = q = 0$. Similarly to the case of non-regular SD, we can compute $\mathsf{C}^R_{\mathsf{ISD}}$. For our parameters selection we consider an estimation of the cost of both this regular variant, associated with the Stern [Ste88] and Finiasz and Sendrier [FS09] techniques, and the more recent non-regular BJMM variant, and then we take the minimum of the two. It would be very interesting to apply more recent techniques to the regular case, but this is out of the scope of this paper. It is worth noting that all the improved ISD variants come at the price of much larger space complexity, which is usually completely ignored in the cost analysis of these algorithms, but plays a significant role in practical applications. For this reason, Prange's plain information set decoding still achieves the best time-space complexity product. However for our parameter selection, as we already said, we consider a lower bound on the cost of BJMM and a special regular-case tailored SD variant, taking into account exclusively the average number of operations and ignoring the memory cost of the algorithm.

C Additional Material for Efficiency Analysis

C.1 BMR Preprocessing: Communication Complexity

Here we detail how we compared our communication complexity with that of the best previous passively secure BMR protocol, namely [BLO16]. To simplify the comparison, we exclude the communication related to input and output wires. Given a circuit C_f with X XOR gates and A AND gates, each of them with fan-in-two and arbitrary fan-out, [BLO16] has the following communication costs:

1. One bit multiplication and $3n$ bit-string multiplications per AND gate, where the strings have length κ. A bit multiplication requires $n(n-1)$ bit-OTs, each of which involves sending $128+2$ bits if instantiated with [IKNP03] or 84 bits if instantiated with [KK13]. Each of the bit-string multiplication can be computed using $n-1$ correlated OTs, at a cost of $128+128$ bits each.
2. Each AND gate has size $4n\kappa$ bits, and each party has a share of it. If the circuit is reconstructed by every party sending her share to P_1, and then P_1 broadcasting the addition of every share, the cost of putting an AND gate together is $8n(n-1)\kappa$ bits.

This gives a total cost of $(130+768+8\kappa)\cdot n\cdot(n-1)\cdot A = 1922\cdot n\cdot(n-1)\cdot A$ bits for the [IKNP03] instantiation and $(67+768+8\kappa)\cdot n\cdot(n-1)\cdot A = 1876\cdot n\cdot(n-1)\cdot A$ when using [DKS+17] instead. In our work, the costs are:

1. One bit multiplication and $3n$ bit/string multiplications per AND gate, where the strings have length ℓ. When implemented with our improved GMW protocol with deterministic committees, these cost $(n-h+1)(n-1)(\ell_{\mathrm{OT}}+\ell_{\mathrm{OT}}\kappa/r+1)$ bits and $n(n-1)\ell_{\mathrm{BMR}}(\ell_{\mathrm{OT}}+\ell_{\mathrm{OT}}\kappa/r+1)$ bits, respectively.
2. Each AND gate has size $4\cdot(n\ell+1)$ bits. Each Splitter gate has size $4n\ell$ bits.

C.2 Instantiating the CRS

Our protocols require a CRS, which is a randomly sampled function, H. One way of implementing this would be generate the function in a setup phase (e.g. with coin-tossing) and store it as a lookup table. However, when the table grows large this will have a prohibitive impact on performance, as there will likely be many cache misses when reading from H at random locations. A more efficient alternative is to implement H using fixed-key AES, which offers fast performance on modern CPUs with AES hardware instructions. This gives security in the ideal cipher model, where fixed-key AES is modelled as a random permutation.[11]

Depending on which of our two protocols is used, this method works as follows:

– For GMW, H is a 1-bit output hash function, so we can simply truncate the AES output.
– For our BMR-style protocol, we need to expand the input to $n\cdot\ell+1$ bits. Let $B=\lceil(n\cdot\ell+1)/128\rceil$ be the number of AES blocks needed to generate one hash output. The parties first fix a random key $s\leftarrow\{0,1\}^{128}$ and then define:

$$\mathsf{H}(i,b,k)=(\mathsf{AES}_s(i\|b\|k\|0),\ldots,\mathsf{AES}_s(i\|b\|k\|B-1)),$$

where the last block is truncated so that the total output length is $n\cdot\ell_{\mathrm{BMR}}+1$ bits.
The cost of a single call to H is that of B AES operations.

[11] This actually only provides security up to the birthday bound, i.e. as long as the adversary makes no more than 2^{64} queries to AES_s. In practice, however, since ℓ is small there will typically be far fewer than 2^{64} possible inputs, so we do not need to be concerned with the birthday bound.