Electronic Colloquium on Computational Complexity, Report No. 108 (2024)

Stochastic Secret Sharing with 1-Bit Shares and Applications to MPC*

Benny Applebaum† Eliran Kachlon†

June 27, 2024

Abstract

The problem of minimizing the share size of threshold secret-sharing schemes is a basic research question that has been extensively studied. Ideally, one strives for schemes in which the share size equals the secret size. While this is achievable for large secrets (Shamir, CACM '79), no similar solutions are known for the case of binary, single-bit secrets. Current approaches often rely on so-called ramp secret sharing that achieves a constant share size at the expense of a slight gap between the privacy and the correctness thresholds. In the case of single-bit shares, this leads to a large gap which is typically unacceptable. The possibility of a meaningful notion of secret sharing scheme with 1-bit shares and almost optimal threshold has been left wide open. Of special interest is the case of threshold 0.5, which is motivated by information-theoretic honest-majority secure multiparty computation (MPC).

In this work, we present a new stochastic model for secret-sharing where each party is corrupted by the adversary with probability p, independently of the other parties, and correctness and privacy are required to hold with high probability over the choice of the corrupt parties. We present new secret sharing schemes with single-bit shares that tolerate any constant corruption probability $p < 0.5$. Our construction is based on a novel connection between such stochastic secret-sharing schemes and error-correcting codes that achieve capacity over the binary erasure channel.

Our schemes are linear and multiplicative. We demonstrate the usefulness of the model by using our new schemes to construct MPC protocols with security against an adversary that passively corrupts an arbitrary subset of $0.499n$ of the parties, where the online communication per party consists of a single bit per AND gate and zero communication per XOR gate. Unlike competing approaches for communication-efficient MPC, our solution is applicable even in a real-time model in which the parties should compute a Boolean circuit whose gates arrive in real-time, one at a time, and are not known in advance.

*This is the full version of a paper published in CRYPTO 2024.

†Tel-Aviv University, Israel bennyap@post.tau.ac.il, elirn.chalon@gmail.com. Supported by ISF grant no. 2805/21 and by the European Union (ERC-2022-ADG) under grant agreement no.101097959 NFITSC.

ISSN 1433-8092

Contents

1 Introduction

Shamir's secret sharing scheme [Sha79] stands out as an ideal n-party threshold secret sharing scheme for several reasons. First, it hides the secret from any set of t parties (*t-privacy*), and allows every $t+1$ parties to recover the secret (*$(t+1)$-correctness*). It is also *linear* over the underlying field $\mathbb{F}$, and the share size is just *one field element*, which is optimal for the secret-domain $\mathbb{F}$. In addition, in the presence of an honest majority (i.e., $t < n/2$) it is also *multiplicative*, which means that if two secret s and s' are shared via the sharing vectors $(s_1, \ldots, s_n)$ and $(s'_1, \ldots, s'_n)$, respectively, then the multiplication $s \cdot s'$ can be computed as a linear combination of the values $s_1 \cdot s'_1, \ldots, s_n \cdot s'_n$. These useful properties make Shamir's scheme an extremely powerful tool with numerous applications, including secure multiparty computation (MPC) (e.g., [BGW88, CCD88, RB89]), cryptographic combiners (e.g., [Her05, HKN+05]) and private information retrieval (e.g., [BI01]).

The main downside of Shamir's scheme is that the underlying field $\mathbb{F}$ must be larger than the number of shares n. In the important case of binary secrets (which is motivated by secure computation of binary circuits), this means that the share size is $\Omega(\log n)$ bits, leading to a "waste" in both communication and storage. Indeed, a dream version of Shamir that works over the binary field would be extremely useful. Unfortunately, coding-theoretic arguments show that such a scheme does not exist. In particular, linear secret sharing schemes with t-privacy and $(t+1)$-correctness are equivalent to *Maximum Distance Separable (MDS)* codes, for which it is known that the alphabet size has to be larger than $(n+2)/2$ for every $1 \leq t \leq n-2$ (see [CDN15]). Furthermore, under standard coding-theoretic conjectures, the alphabet size must be at least $n - O(1)$ [HP03, CDN15].

Known Relaxations. As a next best alternative, previous works slightly relaxed the requirement, and allowed some *gap* between the privacy parameter t_p and the correctness parameter t_c. That is, privacy should hold with respect to every set of size at most t_p, and correctness should hold with respect to any set of size at least t_c, but for any set of size $t \in (t_p, t_c)$ we don't care whether the parties can reveal the secret or not. Such schemes are known as *(t_p, t_c)-ramp secret sharing*, and when the gap is linear, i.e., $t_c - t_p = \epsilon n$ for some constant $\epsilon > 0$, they can be built with a constant alphabet size based on Algebraic Geometric (AG) codes [CC06] or random linear codes [CCG+07]. In the context of MPC, they reduce the share size from logarithmic (as in Shamir) to constant, leading to improved communication complexity at the expense of slightly degrading the resiliency threshold (see, e.g., [CC06]). Ramp secret-sharing has also found applications in other areas of cryptography, such as the construction of efficient zero-knowledge proofs [IKOS09] and OT-combiners [HIKN08]. The downside of these schemes is that the rate degrades with the relative correctness-to-privacy gap ϵ. In particular, in all known constructions of ramp secret sharing the alphabet size grows with $1/\epsilon$, so a small gap between t_p and t_c results in schemes with a large constant-size alphabet. The dependency in $1/\epsilon$ turns to be inherent: it is shown in [BGK20] that the alphabet size should be at least $1/2\epsilon$.

We briefly mention that one can use *packed secret sharing* [FY92, CC06, CCG+07] to share many secrets together using only a single share per party, at the cost of an additional degrading in the resiliency threshold. While this technique allows to improve the rate between the number of shared bits and the share size, there are natural scenarios that do not allow for such a batching. (See Section 1.3.)

This work: 1-bit share size. In this work, we study the question of sharing a *binary secret* with share size of *just one bit*. We ask,

> *Is there a meaningful model that allows for 1-bit Shamir-like secret sharing? Can such a scheme be useful for secure multiparty computation in the standard-model?*

We answer both questions in the affirmative. First, we present a new model, called the *stochastic corruptions model*, were each party is corrupted with probability p, independently of the other parties. For this model, we construct a Shamir-like secret sharing scheme with share size of only one bit for any corruption probability $p < 1/2$. We then show how to transform schemes in this model to schemes in the standard corruptions model, and use this to derive new MPC protocols with improved efficiency.

1.1 Secret Sharing in the Stochastic Corruptions Model

We initiate the study of the *p-stochastic-corruption model*, where the adversary corrupts each party with probability p, independently of the other parties. As a motivating example, consider the basic scenario of n servers that wish to securely compute a joint function of their private inputs, where security is required in case of failures or malfunctions of some of the servers. We can use standard solutions from the MPC literature whose security holds even against a stronger adversary that is allowed to *choose* the identity of the corrupt servers. However, this is an overkill since in our case the failures and malfunctions of the servers behave like a *stochastic process*, and they are not under the control of an adversary. A textbook solution that makes over-pessimistic assumptions on the power of the adversary can still be used, but it may lead to unnecessary overhead.[1] We continue with the definition of secret sharing in the stochastic corruption model.

Stochastic secret sharing. A *p-stochastic secret sharing with error ϵ*, is a randomized algorithm D that maps a secret s to n shares $s_1, \ldots, s_n$, so that the following holds with probability at least $1 - \epsilon$ over the choice of corrupt parties $T \subseteq \{1, \ldots, n\}$ in the p-stochastic corruption model:

- *(Correctness)* Given the shares $(s_i)_{i \in \{1,\ldots,n\} \setminus T}$ of the honest parties, it is possible to recover the secret s.
- *(Privacy)* The shares $(s_i)_{i \in T}$ of the corrupt parties reveal no information about the secret s.

By default, we take ϵ to be negligible in the number of parties. Note that to obtain a sub-constant error probability ϵ, it is required that $p < 1/2$.

1-bit stochastic secret sharing. Quite surprisingly, we show that p-stochastic secret sharing can be realized *with share size of only 1 bit*! The scheme is based on the binary Reed-Muller code (with an appropriate choice of parameters), and the shares are just the evaluations of a random multivariate polynomial over $\mathbb{F}_2$ whose free coefficient is the secret s. In fact, the structure of the Reed-Muller code implies that our secret sharing scheme is also multiplicative, which is an important feature

[1] We note that secret sharing in the stochastic model may also be natural in other areas of cryptography outside of MPC, e.g., in leakage resilient circuits in the random probing model [ISW03], where each wire is leaked to the adversary with probability p, and in combiners to cryptographic primitives whose security is violated with probability p over the internal randomness (see [IMSW14]).

for the applications. The result is summarized in the following theorem. (See Theorem 3.6 for a more detailed statement.)

Theorem 1.1. *There exists an n-party linear secret sharing scheme, such that for every $0 < p < 1/2$ the scheme is a p-stochastic secret sharing scheme with error $\epsilon(n) = 2^{-\Omega_p(\sqrt{n})}$. The scheme has share size of just 1 bit, and the sharing time is $O(n \cdot \log(n))$.[2] In addition, the scheme is multiplicative.*

To obtain the theorem, we establish a strong connection between stochastic secret sharing and linear codes C that, together with their dual $C^{\perp}$, can recover from erasures in the binary erasure channel $\mathsf{BEC}(p)$.[3] We then rely on the breakthrough result of [KKM+17] that shows that Reed-Muller codes achieves capacity over the BEC channel, and exploit the fact that the dual of a Reed-Muller code is also a Reed-Muller code (with different parameters) and so it is also capacity achieving.

Since a random linear code over $\mathbb{F}_2$, as well as its dual, achieve the capacity of BEC with all but negligible probability, we can obtain an alternative to the Reed-Muller construction and still maintain a share size of 1 bit. In fact, using the technique of Druk and Ishai [DI14] we construct a family of linear secret sharing scheme, such that a random scheme from the family is a p-stochastic secret sharing scheme with all but negligible probability, and, in addition, sharing a secret requires only *linear time*. The result is summarized in the following theorem. (See Corollary 3.11 for a more detailed statement.)

Theorem 1.2. *There exists a probabilistic polynomial time algorithm* Gen *that on input 1^n samples the description of an n-party linear secret sharing scheme S_n with 1-bit share size and sharing time $O(n)$,[4] such that for every $0 < p < 1/2$, with probability at least $1 - 2^{-\Omega_p(n)}$ over the choice of S_n by* Gen*, the scheme S_n is p-stochastic secret sharing with error $\epsilon(n) = 2^{-\Omega_p(n)}$.*

In fact, in this construction we can take p to be as large as $p = \frac{1}{2} - \frac{1}{n^{0.49}}$ and still obtain sub-exponential error probability. We also note that for a constant $0 < p < 1/2$ this construction achieves exponentially-small error probability, which is better than the sub-exponential error probability of the Reed-Muller construction. However, unlike the Reed-Muller construction, this construction is non-explicit and, more importantly, not multiplicative. The question of constructing a linear-time multiplicative stochastic secret sharing scheme with share size of 1 bit remains an interesting open problem.[5]

1.2 From Stochastic Corruptions to Standard Corruptions

Perhaps surprisingly, secret sharing schemes in the stochastic model can be successfully used for MPC in the standard corruption model. To show this, let us introduce an intermediate model of *t-static secret sharing* that relaxes the properties of ramp secret sharing without losing the features that are necessary for MPC and other typical applications. We will later see that stochastic secret sharing can be easily upgraded to this model without increasing the share size.

[2] While by default we measure the computational complexity in the RAM model, the sharing procedure in Theorem 1.1 can even be implemented by a *circuit* of size $O(n \cdot \log n)$.

[3] We mention that a similar idea appears implicitly in the work of [IMSW14] in the context of OT-combiners.

[4] In fact, the sharing procedure in Theorem 1.2 can even be implemented by a *circuit* of size $O(n)$.

[5] The construction from Theorem 1.2 can be turned into a multiplicative scheme by using the technique of [CDM00, CCG+07]. However, this transformation increases the share-size to 2 bits and increases the computational complexity to $O(n^2)$. (The exponentially-small error probability is preserved). See Remark 3.12 for more details.

t-static secret sharing. A t-static secret sharing scheme with error ϵ, is a secret sharing scheme with *public randomness* r. It allows a dealer to share a secret s among n parties, so that for every set of corrupt parties $T \subseteq \{1, \ldots, n\}$ of size at most t, the following holds with probability at least $1 - \epsilon$ over the choice of the public randomness r:

- *(Correctness)* The honest parties (i.e., the parties in $\{1, \ldots, n\} \setminus T$) can recover the secret s.
- *(Privacy)* The corrupt parties in T have no information about the secret s.

We emphasize that the public randomness r is chosen *once and for all*, and can be reused by the dealer, and even by other dealers. However, it is crucially assumed that the adversary chooses which parties to corrupt independently of r, and so this notion is tailored to the case of static adversaries. Indeed, in the common setting of MPC with static security, one needs only correctness with respect to the fixed set of honest parties and privacy with respect to the fixed set of corrupted parties. In this setting, the properties of standard secret sharing (i.e., correctness/privacy for *all* large/small subsets) are simply an "overkill"!

From stochastic corruptions to t corruptions. It is not hard to transform any p-stochastic secret sharing S into a t-static secret sharing S', where $t = \lfloor pn \rfloor$, while preserving the share size of S. To do so, we simply permute the identity of the parties. That is, the public randomness of S' consists of a public random permutation π of $\{1, \ldots, n\}$, and to share a secret s among the n parties, S' first samples the shares $(s_1, \ldots, s_n)$ according to S and sends the share $s_{\pi(i)}$ to the ith party.

We think about permuting the parties as a preprocessing step, that is executed once and for all, and can be computed in time $O(n \cdot \log(n))$. Jumping ahead, in the context of MPC this preprocessing step can be implemented at the beginning of the protocol by executing a sub-protocol for permuting the identities. Once the permutation is sampled, we can share a secret under S' with complexity that is linear in the sharing complexity of S.

We therefore obtain the following corollaries. (See Theorem 4.5 and Theorem 4.6 for more detailed versions.)

Corollary 1.3. *There exists an n-party linear secret sharing scheme, such that for every $0 < p < 1/2$ the scheme is pn-static secret sharing scheme with error $\epsilon(n) = 2^{-\Omega_p(\sqrt{n})}$. The scheme has share size of just 1 bit and public randomness of $O(n \cdot \log(n))$ bits. In addition, the scheme is multiplicative, and sharing a secret requires $O(n \cdot \log(n))$ operations in the RAM model.*

Corollary 1.4. *There exist a probabilistic polynomial time algorithm* Gen *that on input 1^n samples the description of an n-party secret sharing scheme S_n with 1-bit share size, such that for every $0 < p < 1/2$, with probability at least $1 - 2^{-\Omega_p(n)}$ over the choice of S_n by* Gen*, the scheme S_n is pn-static linear secret sharing scheme with error $\epsilon(n) = 2^{-\Omega_p(n)}$. In addition, after a preprocessing time of $O(n \cdot \log(n))$, sharing a secret requires $O(n)$ time in the RAM model.*

Like in the case of Theorem 1.2, in Corollary 1.4 we can take p to be as large as $p = \frac{1}{2} - \frac{1}{n^{0.49}}$ and still obtain sub-exponential error probability.[6]

[6]We note that achieving a threshold of $\frac{1}{2} - O(\frac{1}{n})$ is impossible. See Section 4.2 for more details.

Comparison with ramp secret sharing. We observe that for every $t \leq \frac{1-\epsilon}{2} \cdot n$, a $(t, t+\epsilon n)$-ramp secret sharing is also a t-static secret sharing with zero error and no public randomness. (Indeed, privacy holds for every set of t corrupt parties, while correctness holds for the set of honest parties since it is of size at least $n - t \geq t + \epsilon n$.) The converse direction does not hold since ramp secret sharing provides security against an adaptive adversary. Overall, t-static secret sharing is a strict relaxation of ramp secret sharing. This relaxation is, in fact, *necessary* for obtaining share size of 1 bit, since it is known [BGK20] that any (t_p, t_c)-ramp secret sharing with share size of 1 bit must satisfy $t_c - t_p \geq (n+2)/3$. Put differently, a $(t, t+\epsilon n)$-ramp secret sharing with single-bit shares must satisfy $\epsilon \geq 1/3$. In comparison, our t-static secret sharing schemes achieve 1-bit share size for t arbitrarily close to $n/2$. In Section A we compare our scheme to concrete constructions of ramp secret sharing from AG codes and random linear codes.

1.3 Application: Efficient MPC in the Real-Time Computation Model

Using the preprocessing techniques of [BH08], our secret-sharing schemes can be used to realize communication-efficient MPC protocols for computing a Boolean circuit C, in which each party communicates a single bit per gate. In fact, we can even obtain a preprocessing phase in which the communication complexity per party is $O(|C|)$, where $|C|$ is the size of C.

Theorem 1.5 (Informal). *Let $\epsilon > 0$, let n be the number of parties, and let t be number of corrupt servers such that $t = (\frac{1}{2} - \epsilon)n$. Then for every Boolean circuit C there exists a statistically-secure n-party MPC protocol with error $2^{-\Omega_\epsilon(\sqrt{n})}$ that securely computes C. The protocol has a preprocessing phase that depends only on $|C|$ and has communication $O(n \cdot |C|)$. In addition, in the online phase every party communicates a single bit per gate.*

As part of our construction we provide a seedless randomness extractor that takes a binary string $\mathbf{x} \in \{0,1\}^n$ that has $(\frac{1}{2} + \epsilon)n$ random bits *in random coordinates* $I \subseteq \{1, \ldots, n\}$, and with probability $2^{-\Omega_\epsilon(n)}$ over the choice of I outputs a random binary string $\mathbf{y}$ of length $(\frac{1}{2} + \frac{\epsilon}{2})n$. This can be seen as a variant of the bit-extraction problem of [CGH+85]. Using the technique of [DI14], our randomness extractor can even be described as a linear function with *linear-size circuit*. We mention that previous works could achieve similar results only for a larger alphabet size, or require $\mathbf{x}$ to have a much larger fraction of random bits (see, e.g., [BH08, CCXY18, LXYY23]).

Comparison with previous works. Most information-theoretic secure computation protocols either rely on gate-by-gate sharing or on packed secret sharing.[7] The first, more classical approach [BGW88, CCD88], allocates a single share per each wire of the circuit C. In an online/offline setting this *gate-by-gate* approach leads to online communication of a single share per party for each multiplication gate, and so the total online communication is at most $|C|n \cdot k$ where k is the bit length of shares. Our solution fits into this framework and achieves $k = 1$ and privacy threshold of 0.499. Previous gate-by-gate solutions suffer from worse tradeoffs that are inherited from the the aforementioned size-vs-privacy tradeoffs in ramp secret sharing schemes. A more modern approach [DIK10, GIP15, GIOZ17, GPS21] bypasses the gate-by-gate paradigm by

[7] One can further reduce communication by delegating the computation to a small random committee of super-logarithmic size. This technique can be applied on top of the two other approaches and so we ignore it in the current discussion. We do mention that this approach reduces the overall communication but typically incurs a relatively large (e.g., quasi-polynomial) error probability for the event that the adversary corrupts a majority of the committee.

packing together several gates into a single share via the use of packed secret sharing [FY92]. In this way, the parties can securely compute a block of ℓ gates at the cost of a single computation. Combining this technique with additional ideas, [GPS21] have constructed a protocol with communication overhead of $O(|C|)$ bits (assuming that the circuit is larger than the number of inputs and outputs). However, packed secret sharing has several drawbacks. First, for every layer it requires manipulating the order of the packed secrets, which result in a large computation overhead. Second, it requires knowing *in advance* the structure of the next layer of computation in the circuit. Therefore, it seems inapplicable in the case where the circuit is not known in advance, and the gates are revealed in real-time, one-at-a-time. In comparison, our protocol is very efficient: the total computation overhead in the preprocessing phase is $O(n \cdot \log(n))$ while the total computation overhead in the online phase is merely $O(n)$ Boolean operations. In addition, the preprocessing of our protocol does not depend on the computed circuit, but only on its size, and therefore our protocol can handle the gates one-by-one in an online manner. This is especially useful in a *real-time computation model* where the gates of the circuit are chosen dynamically on-the-fly. Details follow.

1.3.1 The Real-Time Computation Model

Let us begin with two motivating examples. Assume that an ordered array, that contains m distinct keys (integers), is secret shared among n servers. The servers also hold a secret sharing of a key k^* that appears in the array, and the goal of the servers is to learn the rank (i.e., the location) of k^* in the array, without learning any other information about the array. The classical approach is to consider a circuit that implements the search for k^*, and securely compute this circuit. However, this circuit depends on the entire array, so its size is at least linear in m. We note that a better approach would be if we do not use the circuit model, which is oblivious, and run a sequence of operations that is chosen on the fly based on the results of previous operations. Concretely, the servers can simply apply binary search algorithm, with secure comparisons at each step. That is, whenever the binary search algorithm requires comparing k^* to the ith key in the array, the servers engage in a secure protocol to check whether k^* is smaller, larger, or equal to the ith key. Given this information, the servers can now continue with the binary search, until they find the location of k^* in the array. This reduces the dependency on m from linear (in the circuit-based approach) to logarithmic, and also preserves security, as the results of the comparisons can be deduced from the rank of k^*, so the servers learn no information other than the rank.

Let us further consider a scenario in which several servers hold some shared state S of a function F (e.g., a Large Language Model) and a client streams public inputs $p_1, \ldots, p_i, \ldots$ to F that should lead to an online sequence of (possibly public) outputs $y_1, \ldots, y_i, \ldots$ and to updates in the state. We can translate each bit p_i to a sequence of Boolean gates that should be applied on the shared state. In this case, these gates are not known at the beginning and are only available during run-time. Moreover, the computation is reactive and the bit p_{i+1} may depend on previous outcomes $y_1, \ldots, y_i$.

The real-time computation model. These examples demonstrate that in some scenarios it is natural to choose which instructions to perform based on intermediate values of the computation. We consider a new model of computation, called the *real-time computation model*. The model consists of a single client and n servers, and there are m input-bits $x_1, \ldots, x_m \in \{0, 1\}$ that are secret shared among the servers. The goal is to allow the client to guide the servers in the manipulation of the

shared inputs, without letting the servers learn any information about the inputs.

More formally, the client has access to an interface that includes $R \geq m$ registers, where the ith input bit is stored in the ith register. In each step the client can ask (1) to sum/multiply the content of registers i and j and put the result in register k, (2) to assign some value to a register, (3) to increase the number of registers, and (4) to reveal the value of the ith register to the client. The servers get the instructions from the client one-by-one and should perform operations over the secret-shared values, so that when the client asks to reveal the value of the ith register, the servers can send the client information that allows her to compute this value. We formalize this model in the framework of universal composability (UC) in Section 5, using an ideal functionality.

We note that basic protocols in the real-time model can be constructed based on the BGW-paradigm [BGW88] assuming that $t < n/2$, and even the GMW-paradigm [GMW87] for any $t \leq n$. In Section 5 we then prove that Theorem 1.5 holds even in the real-time computation model. In particular, for each Boolean instruction each server sends at most a single bit which seems to be optimal in the real-time model. (See [DNPR16] for a lower-bound in a related model.)

Organization. The paper is organised as follows. In Section 2 we present preliminaries regarding error correcting codes and secret sharing. In Section 3 we discuss stochastic secret sharing, present a general transformation from error-correcting codes to stochastic secret sharing, and provide constructions. Then, in Section 4 we discuss t-static secret sharing, show how to transform stochastic secret sharing to t-static secret sharing, and prove a lower bound on 1-bit static secret sharing. In Section 5 we present applications to MPC. In the appendix, a comparison of our codes with concrete constructions of ramp secret sharing appears in Section A.

2 Preliminaries

In this section we present basic preliminaries regarding error correcting codes, secret sharing and probability theory.

2.1 Error Correcting Codes

We briefly recall basic notions in error correcting codes. For a more detailed explanation, the reader is referred to [Rot06, GRS23]. Throughout, we take any vector $\mathbf{x}$ to be a row vector, and denote the corresponding column vector by $\mathbf{x}^T$.

Basics. For a prime power q and positive integers $k \leq n$, an $[n,k]_q$ code C is a linear subspace of $\mathbb{F}_q^n$ of dimension k. We denote the rate of the code by $\rho := k/n$. A *generator matrix* of C is a $k \times n$ matrix $G \in \mathbb{F}_q^{k \times n}$, whose rows span C. The elements of C are called codewords, and every codeword $\mathbf{c}$ can be written (uniquely) as $\mathbf{c} = \mathbf{m} \cdot G$ for some vector $\mathbf{m} \in \mathbb{F}_q^k$. There exists an $(n-k) \times n$ matrix $H \in \mathbb{F}_q^{(n-k) \times n}$, called the *parity check matrix*, that satisfies $H \cdot \mathbf{y}^T = 0$ if and only if $\mathbf{y} \in C$. The dual code of C, denoted $C^\perp$, is the $[n, n-k]_q$ code generated by the parity check matrix H.

Erasure channels and MAP-decoder. For $0 < p < 1$, the *q-ary erasure channel with erasure probability* p, denoted $\mathsf{QEC}(p)$, takes as an input a vector $\mathbf{x} \in \mathbb{F}_q^n$ and outputs a vector $\mathbf{y} \in (\mathbb{F}_q \cup \{?\})^n$

that is obtained from $\mathbf{x}$ by replacing the ith entry of $\mathbf{x}$ with the erasure symbol '?' with probability p, independently of the other coordinates. The *binary erasure channel with erasure probability* p, denoted $\mathsf{BEC}(p)$, is just the 2-ary erasure channel with erasure probability p. We usually denote the set of erased coordinates by $J \subseteq [n]$, and the set of non-erased coordinates by $I = \{1, \ldots, n\} \setminus J$.

For an $[n, k]_q$ code C, we define the *block-maximum-a-priori decoder* (block-MAP decoder) of C to be the decoder that (1) takes as an input a vector $\mathbf{y} \in (\mathbb{F}_q \cup \{?\})^n$, and (2) if there exists a unique codeword $\mathbf{c} \in C$ that agrees with $\mathbf{y}$ on all non-erased coordinates, the decoder returns $\mathbf{c}$; otherwise the decoder returns a special failure symbol $\perp$. We note that for linear codes, the block-MAP decoder boils down to finding a unique solution of a linear system of equations or announcing that no unique solution exists, which can be done in polynomial time via Gaussian Elimination.

Remark 2.1 (Decoding from erasures for linear codes.)**.** *For a linear $[n, k]_q$ code C, the block-MAP decoder successfully recovers a codeword $\mathbf{c} \in C$ that was transmitted through the q-ary erasure channel if and only if the set of erased coordinates $J \subseteq \{1, \ldots, n\}$ does not cover any codeword, i.e., if there is no non-zero codeword $\mathbf{c}' \in C$ such that $\mathbf{c}'[i] \neq 0$ only if $i \in J$. In particular, the success of the block-MAP decoder does not depend on the choice of the codeword $\mathbf{c}$, but only on the erasure pattern J.*

Indeed, if the codeword $\mathbf{c}$ was transmitted and the vector $\mathbf{y}$ is the received word, then the block-MAP decoder fails if and only if there exists another codeword $\mathbf{c}'' \in C$ that agrees with $\mathbf{c}$ on all non-erased coordinates. Therefore, the codeword $\mathbf{c}' := \mathbf{c} - \mathbf{c}''$ is zero on all entries outside J, and it is non-zero since $\mathbf{c}$ and $\mathbf{c}'$ disagree on at least one coordinate in J. For the other direction, note that if there exists a non-zero codeword $\mathbf{c}' \in C$ such that $\mathbf{c}'[i] \neq 0$ only if $i \in J$, then the codeword $\mathbf{c}'' = \mathbf{c} + \mathbf{c}'$ agrees with $\mathbf{c}$ on all coordinates outside J, and disagrees with $\mathbf{c}$ on at least one coordinate in J, so the block-MAP decoder fails.

Asymptotic complexity. To capture the asymptotic complexity and parameters of the codes we assume that there exists a deterministic polynomial time algorithm Gen that on input 1^n outputs a description of the encoding and decoding function of a code C_n of length n (e.g., circuits or a RAM programs that compute the encoding and decoding). In most cases we are interested in binary codes C_n that together with their dual codes achieve negligible error probability $\epsilon(n)$ over the erasure channel $\mathsf{BEC}(p)$ for some given constant p.

We are also interested in the case where Gen is allowed to be probabilistic, so on input 1^n the algorithm Gen samples a description of the encoding and decoding functions of a code C_n from a family $\mathcal{S}_n$ of codes of length n. In this case we usually require that, except with negligible probability, with overwhelming probability $1 - n^{-\omega(1)}$ over the internal randomness of Gen, the code C_n satisfies some property, e.g., together with its dual code it achieves negligible error probability $\epsilon(n)$ in the erasure channel $\mathsf{BEC}(p)$.

2.2 Secret Sharing

We continue with basic notions in secret sharing. The following definitions are taken with minor changes from [ANP23].

Definition 2.2 (Partial access structure)**.** *A* partial access structure *over n parties is a pair (Γ_0, Γ_1) where $\Gamma_0, \Gamma_1 \subseteq 2^{[n]}$ are non-empty collections of sets such that (1) $B \not\subseteq A$ for every $A \in \Gamma_0$, $B \in \Gamma_1$, (2) for every $A \in \Gamma_0$ and $B \subseteq A$ it holds that $B \in \Gamma_0$, and (3) for every $A \in \Gamma_1$ and $B \supseteq A$ it holds that $B \in \Gamma_1$. Sets in Γ_1 are called* authorized*, and sets in Γ_0 are called* unauthorized*.*

*For positive integers $t_p < t_c \leq n$ the (t_p, t_c)-*ramp access structure *over n parties (Γ_0, Γ_1) is defined by letting Γ_0 be the collection of all subsets of size at most t_p and letting Γ_1 be the collection of all subsets of size at least t_c.*

Definition 2.3 (Secret sharing and ramp secret sharing)**.** *Let $\mathcal{S}$ be a finite set of size at least 2, let $\mathcal{R}$ be a finite set, and let (Γ_0, Γ_1) be a partial access structure. An n-party secret sharing scheme that realizes the partial access structure (Γ_0, Γ_1) with domain of secrets $\mathcal{S}$ and randomness domain $\mathcal{R}$, is a pair of algorithms* $(\mathsf{Share}, \mathsf{Recover})$ *such that*

- Share *is a randomized algorithm, that takes a secret $s \in \mathcal{S}$ and randomness $r \in \mathcal{R}$, and returns n shares $s_1, \ldots, s_n$.*
- $\mathsf{Recover}$ *is a deterministic algorithm, that takes a set $I \subseteq \{1, \ldots, n\}$ and shares $(s_i)_{i \in I}$ and either returns some element $s' \in \mathcal{S}$ or a failure symbol $\perp$.*

The algorithms satisfy the following properties.

- (Correctness) *For every secret $s \in \mathcal{S}$, every fixed randomness $r \in \mathcal{R}$, and every authorized set $I \in \Gamma_1$, it holds that $\mathsf{Recover}(I, (s_i)_{i \in I}) = s$, where $(s_1, \ldots, s_n) = \mathsf{Share}(s; r)$.*
- (Privacy) *For every unauthorized set $I \in \Gamma_0$, and for every pair of secrets $s, s' \in \mathcal{S}$, it holds that the random variables $(s_i)_{i \in I}$ have the same distribution as the random variables $(s'_i)_{i \in I}$, where $(s_1, \ldots, s_n) = \mathsf{Share}(s; r)$, $(s'_1, \ldots, s'_n) = \mathsf{Share}(s'; r)$, and r is uniformly distributed over $\mathcal{R}$.*

*For positive integers $t_p < t_c \leq n$, an n-party (t_p, t_c)-*ramp secret sharing scheme *with domain of secrets $\mathcal{S}$ and randomness domain $\mathcal{R}$ is a secret sharing scheme that realizes the (t_p, t_c)-ramp access structure.*

A t-out-of-n secret sharing scheme is a $(t-1, t)$-ramp secret sharing scheme. A secret sharing scheme is *linear* over a finite field $\mathbb{F}_q$ if the secret-domain $\mathcal{S}$ equals to $\mathbb{F}_q$, the randomness-domain $\mathcal{R}$ is a vector space over $\mathbb{F}_q$, and the function Share is a linear transformation over $\mathbb{F}_q$. We will restrict our attention to linear secret sharing schemes in which each share is a *single* field element. Note that, by linearity, if $(s_1, \ldots, s_n)$ is a valid sharing of a secret s and $(s'_1, \ldots, s'_n)$ is a valid sharing of s' then, for any scalars $a, b \in \mathbb{F}_q$, it holds that $(a \cdot s_1 + b \cdot s'_1, \ldots, a \cdot s_n + b \cdot s'_n)$ is a valid sharing of $a \cdot s + b \cdot s'$.

A linear secret sharing scheme is *multiplicative* [CDM00] if for every secret s (resp., s') and every valid shares $(s_1, \ldots, s_n)$ (resp., $(s'_1, \ldots, s'_n)$), it holds that $s \cdot s'$ can be written as a linear combination of $s_1 \cdot s'_1, \ldots, s_n \cdot s'_n$. A t-out-of-n secret sharing scheme is *strongly multiplicative* if for any set I of size at least $n - (t-1)$, it holds that $s \cdot s'$ can be written as a linear combination of $(s_i \cdot s'_i)_{i \in I}$.

To capture the asymptotic complexity of the scheme, we think of $t_p(n)$ and $t_c(n)$ as functions of n, and consider a deterministic algorithm Gen that on input 1^n outputs the description of the algorithms $(\mathsf{Share}_n, \mathsf{Recover}_n)$ of an n-party $(t_p(n), t_c(n))$-ramp secret sharing scheme. In some cases we allow Gen to be probabilistic, and then we require that with overwhelming probability over the internal randomness of Gen, the output $(\mathsf{Share}_n, \mathsf{Recover}_n)$ of Gen is an n-party $(t_p(n), t_c(n))$-ramp secret sharing scheme.

2.3 A Useful Fact

We will use the following simple variant of Markov's inequality.

Fact 2.4. *Let A and B be non-empty sets, and let $P : A \times B \to \{0,1\}$ be a predicate. Let $\mathcal{A}$ be any distribution over A, and let $\mathcal{B}$ be an independent distribution over B. Assume that*

$$\Pr_{\substack{a\leftarrow\mathcal{A}\\ b\leftarrow\mathcal{B}}}[P(a,b)=0] \leq \epsilon$$

for some $\epsilon > 0$. Then with probability at least $1-\sqrt{\epsilon}$ over the choice of $a \leftarrow \mathcal{A}$ it holds that

$$\Pr_{b\leftarrow\mathcal{B}}[P(a,b)=0] \leq \sqrt{\epsilon}. \tag{1}$$

Proof. Say that $a \in A$ is "good" if it satisfies (1), and otherwise say that it is "bad". Our goal is to prove that $a \leftarrow \mathcal{A}$ is good with probability at least $1-\sqrt{\epsilon}$. Assume towards contradiction that $a \leftarrow \mathcal{A}$ is bad with probability more than $\sqrt{\epsilon}$. Then

$$\epsilon \geq \Pr_{\substack{a\leftarrow\mathcal{A}\\ b\leftarrow\mathcal{B}}}[P(a,b)=0] \geq \Pr_{\substack{a\leftarrow\mathcal{A}\\ b\leftarrow\mathcal{B}}}[P(a,b)=0 \mid a \text{ is bad}] \cdot \Pr_{a\leftarrow\mathcal{A}}[a \text{ is bad}] > \sqrt{\epsilon}\cdot\sqrt{\epsilon} = \epsilon,$$

in contradiction. The fact follows. □

3 Stochastic Secret sharing

In this section we construct stochastic secret sharing schemes. In Section 3.1 we prove that every error-correcting code that, together with its dual code, perform well over the erasure channel, can be transformed into a stochastic secret sharing scheme. Then, we use this transformation to construct a stochastic secret sharing scheme from the Reed-Muller code (Section 3.2) and from linear codes with linear-size circuit (Section 3.3). Whenever possible, we present our results in a general way, over an arbitrary finite field $\mathbb{F}_q$. Our final constructions are then obtained by taking $q = 2$.

We begin with a formal definition of stochastic secret sharing.

Definition 3.1 (p-stochastic secret sharing)**.** *Let $0 < p < 1/2$ and $0 < \epsilon < 1$. Let $n > 0$ be an integer, and let $\mathcal{S}$ be a finite set of size at least 2. An n-party p-stochastic secret sharing scheme wtih error ϵ for a domain of secrets $\mathcal{S}$ is a pair of algorithms* $(\mathsf{Share}, \mathsf{Recover})$ *such that*

- Share *is a randomized algorithm, that takes a secret $s \in \mathcal{S}$ and randomness, and returns n shares $s_1, \ldots, s_n$.*
- $\mathsf{Recover}$ *is a deterministic algorithm, that takes a set $I \subseteq \{1,\ldots,n\}$ and shares $(s_i)_{i\in I}$ and either returns some element $s' \in \mathcal{S}$ or a failure symbol $\perp$.*

The algorithms satisfy the following properties. Let $\mathsf{Corrupt}$ *be the random variable corresponding to the set of corrupt parties, where every $i \in \{1,\ldots,n\}$ belongs to* $\mathsf{Corrupt}$ *with probability p, independently of the other parties. Let* $\mathsf{Honest} = \{1,\ldots,n\} \setminus \mathsf{Corrupt}$ *be the set of honest parties. Then with probability $1-\epsilon$ over the choice of* $\mathsf{Corrupt}$*, the following properties hold:*

- (Correctness) *For every secret $s \in \mathcal{S}$, and every string r of randomness, it holds that* $\mathsf{Recover}(\mathsf{Honest}, (s_i)_{i\in\mathsf{Honest}}) = s$*, where* $(s_1,\ldots,s_n) = \mathsf{Share}(s;r)$*.*

- (Privacy) *For every pair of secrets $s, s' \in \mathcal{S}$, it holds that the random variables $(s_i)_{i\in\mathsf{Corrupt}}$ have the same distribution as the random variables $(s'_i)_{i\in\mathsf{Corrupt}}$, where $(s_1, \ldots, s_n)$ are sampled from $\mathsf{Share}(s; r)$ and $(s'_1, \ldots, s'_n)$ are sampled from $\mathsf{Share}(s'; r)$, and r is uniformly distributed.*

Equivalently, a secret sharing scheme that realizes an n-party partial access structure (Γ_0, Γ_1) is a p-stochastic secret sharing scheme with error ϵ if

$$\Pr[\mathsf{Corrupt} \in \Gamma_0 \bigwedge \mathsf{Honest} \in \Gamma_1] \geq 1 - \epsilon,$$

where $\mathsf{Corrupt} \subset [n]$ is sampled by choosing each $i \in [n]$ independently with probability p and $\mathsf{Honest} = \{1, \ldots, n\} \setminus \mathsf{Corrupt}$.

We note that the definitions of linear and multiplicative secret sharing schemes extend naturally to stochastic secret sharing. For the asymptotic notion, we consider a (possibly probabilistic) algorithm Gen that on input 1^n outputs the description of the algorithms $(\mathsf{Share}_n, \mathsf{Recover}_n)$ of an n-party secret sharing scheme. We usually require that for every constant $0 < p < 1/2$, with overwhelming probability over the internal randomness of Gen, the scheme is p-stochastic secret sharing with negligible error $\epsilon_p(n)$.

3.1 From Erasure Coding to Stochastic Secret Sharing

We begin with the following standard transformation from error-correcting codes to secret sharing schemes [MS81, Mas93, Mas95].

Construction 3.2. *Let $\mathbb{F}_q$ be a finite field, and let C be a linear error-correcting code over $\mathbb{F}_q$ with block length $n + 1$. Define $\mathsf{StochSS}(C)$ in the following way:*

- (Sharing) *The randomized algorithm Share takes as an input a secret $s \in \mathbb{F}_q$, picks a random codeword $(c_0, c_1 \ldots, c_n) \leftarrow C$ conditioned on $c_0 = s$, sets $s_i := c_i$ for every $i \in \{1, \ldots, n\}$ and outputs the shares $(s_1, \ldots, s_n)$.*

- (Recovery) *The algorithm $\mathsf{Recover}$ takes as an input a set $I \subseteq \{1, \ldots, n\}$ and shares $(s_i)_{i\in I}$. If there exists a unique codeword $\mathbf{c} \in C$ such that $\mathbf{c}[i] = s_i$ for all $i \in I$, the algorithm returns $\mathbf{c}[0]$. Otherwise, the algorithm returns a failure symbol $\perp$.*

Modern works (e.g., [CCG+07, Theorem 1]) have shown that if C and its dual $C^\perp$ behave well with respect to *worst-case* erasures (i.e., both of them have a good distance), then the resulting secret sharing scheme is a ramp secret sharing scheme. The following lemma shows that in order to get stochastic secret sharing it is enough to require that C and $C^\perp$ behave well with respect to *stochastic erasures* in $\mathsf{QEC}(p)$.

Lemma 3.3 (Main lemma)**.** *Let $n > 0$ be an integer, let $\mathbb{F}_q$ be a finite field, let C be a linear error-correcting code over $\mathbb{F}_q$ with block length $n + 1$, and let $C^\perp$ be its dual code. Let $0 < p < 1/2$ and assume that*

1. *The probability that the block-MAP decoder of C fails in the q-ary erasures channel $\mathsf{QEC}(p)$ is at most ϵ, and,*

2. *The probability that the block-MAP decoder of $C^\perp$ fails in the q-ary erasures channel $\mathsf{QEC}(p)$ is at most δ.*

Then $\mathsf{StochSS}(C)$ is a linear n-party p-stochastic secret sharing scheme with error $(\epsilon + \delta)/p$.

Proof. In the following we denote by $J \subseteq \{0, 1, \ldots, n\}$ the random variable that corresponds to the set of coordinates erased by the $\mathsf{QEC}(p)$ channel, and by $I = \{0, 1, \ldots, n\} \setminus J$ the set of non-erased coordinates. We also denote by $\mathsf{Corrupt} \subseteq \{1, \ldots, n\}$ the random variable corresponding to the set of corrupt parties in the p-stochastic corruptions model, and the set of honest parties by $\mathsf{Honest} = \{1, \ldots, n\} \setminus \mathsf{Corrupt}$. We observe that the random variable Honest has the same distribution as the random variable I conditioned on the event that $0 \notin I$; Equivalently, the random variable $\mathsf{Corrupt} \cup \{0\}$ has the same distribution as the random variable J conditioned on the event that $0 \in J$.

Let $\mathcal{D}$ be the block-MAP decoder of C, and let $\mathcal{D}^\perp$ the block-MAP decoder of $C^\perp$. Recall that $\mathcal{D}$ (resp., $\mathcal{D}^\perp$) fails if and only if the set of erased coordinates J covers some codeword in C (resp., $C^\perp$). (See Remark 2.1.) We continue with the proof of correctness.

Correctness. We first prove that correctness holds with probability $1 - \epsilon/p$ over the choice of $\mathsf{Corrupt}$. We observe that correctness holds whenever $\mathcal{D}$ succeeds recovering from erasures when the non-erased coordinates are Honest. Indeed, in this case the honest parties can recover the codeword $\mathbf{c}$ and output the secret c_0. By assumption

$$\begin{aligned}
\epsilon &\geq \Pr_I[\mathcal{D} \text{ fails when non-erased coordinates are } I] \\
&\geq \Pr_I[\mathcal{D} \text{ fails when non-erased coordinates are } I \mid 0 \notin I] \cdot \Pr[0 \notin I] \\
&= \Pr_{\mathsf{Honest}}[\mathcal{D} \text{ fails when non-erased coordinates are } \mathsf{Honest}] \cdot p,
\end{aligned}$$

and therefore, the probability that $\mathcal{D}$ fails when the non-erased coordinates are Honest is at most ϵ/p.

Privacy. Next, we prove that privacy holds with probability $1 - \delta/p$ over the choice of $\mathsf{Corrupt}$. Let $G \in \mathbb{F}_q^{k \times (n+1)}$ be the generator matrix of C. We first observe that if the columns of G when restricted to $\mathsf{Corrupt} \cup \{0\}$ are linearly independent, then privacy holds. Indeed, in this case the restriction of a random codeword to the columns in $\mathsf{Corrupt} \cup \{0\}$ results in a random vector, and therefore, even conditioned on $c_0 = s$, the shares of the corrupt parties are uniformly distributed.

Therefore, it remains to prove that the columns in $\mathsf{Corrupt} \cup \{0\}$ are linearly independent with probability at least $1 - \delta/p$. Observe that if the columns in $\mathsf{Corrupt} \cup \{0\}$ are linearly dependent if and only if there exists a non-zero vector $\mathbf{c} \in \mathbb{F}_q^n$ such that (1) $G \cdot \mathbf{c}^T = 0$, and (2) $\mathbf{c}[i] \neq 0$ only if $i \in \mathsf{Corrupt} \cup \{0\}$. Since G is the parity check matrix of $C^\perp$, this occurs if and only if there exists a non-zero codeword $\mathbf{c} \in C^\perp$ such that $\mathbf{c}[i] \neq 0$ only if $i \in \mathsf{Corrupt} \cup \{0\}$, and as we've noted, this occurs if and only if $\mathcal{D}^\perp$ fails when the erased coordinates are $\mathsf{Corrupt} \cup \{0\}$. Since the random variable $\mathsf{Corrupt} \cup \{0\}$ is distributed like the random variable J conditioned on the event that $0 \in J$, we obtain,

$$\begin{aligned}
\delta &\geq \Pr_J[\mathcal{D}^\perp \text{ fails when erased coordinates are } J] \\
&\geq \Pr_J[\mathcal{D}^\perp \text{ fails when erased coordinates are } J \mid 0 \in J] \cdot \Pr[0 \in J] \\
&= \Pr_{\mathsf{Corrupt}}[\mathcal{D}^\perp \text{ fails when erased coordinates are } \mathsf{Corrupt} \cup \{0\}] \cdot p,
\end{aligned}$$

and therefore, the columns in $\mathsf{Corrupt} \cup \{0\}$ are linearly independent with probability at least $1 - \delta/p$.

Conclusion. We conclude that with probability $1-\epsilon/p-\delta/p$, both correctness and privacy hold, as required. In addition, it is not hard to see that the scheme is linear. This concludes the proof of the lemma. □

3.2 Stochastic Secret Sharing from Binary Reed-Muller Code

In this section we prove Theorem 1.1. For an m-variate polynomial $f(x_1,\ldots,x_m) \in \mathbb{F}_2[x_1,\ldots,x_m]$, denote by $\mathsf{Eval}(f) \in \{0,1\}^{2^m}$ the vector of evaluations of f on all the elements of $\mathbb{F}_2^m$. For integeres $0 \leq r \leq m$, we define the (binary) Reed-Muller code with parameters r, m as follows:

$$\mathsf{RM}(r,m) = \{\mathsf{Eval}(f) : f \in \mathbb{F}_2[x_1,\ldots,x_m], \quad \deg(f) \leq r\}.$$

It is well-known (see, e.g., [GRS23]) that the length of the code is $n = 2^m$, the dimension is $k = \sum_{i=0}^{r}\binom{m}{i}$, the distance is 2^{m-r}, and that the dual of $\mathsf{RM}(r,m)$ is $\mathsf{RM}(m-r-1,m)$.

The following theorem, which is a special case of the results of [KKM+17], shows that $\mathsf{RM}(\lfloor m/2 \rfloor, m)$ achieves the capacity of the binary erasure channel.

Theorem 3.4 ([KKM+17])**.** *For every constant $0 < p < 1/2$ there exists a constant $\delta > 0$ such that for sufficiently large m, the error probability of the block-MAP decoder of $\mathsf{RM}(\lfloor m/2 \rfloor, m)$ in $\mathsf{BEC}(p)$ is at most $\exp(-\delta \cdot 2^{m/2})$.*

For every $m > 1$ let C_m be the code $\mathsf{RM}(\lfloor (m-1)/2 \rfloor, m)$, and observe that the length of C_m is 2^m. We continue with the definition of our secret sharing scheme.

Construction 3.5. *For every positive integer n we define the n-party secret sharing scheme $\mathsf{RMStochSS}_n$ with domain of secrets $\{0,1\}$ in the following way.*

- *If $n = 2^m - 1$ for some positive integer m, we let $\mathsf{RMStochSS}_n$ be the n-party secret sharing scheme defined by applying Construction 3.2 on the code C_m.*
- *Otherwise, if $2^m - 1 < n < 2^{m+1} - 1$ for some positive integer m, we define $\mathsf{RMStochSS}_n$ to be the scheme that executes $\mathsf{RMStochSS}_{2^m-1}$ with the first $2^m - 1$ parties. That is, to share a secret $s \in \{0,1\}$ the first $2^m - 1$ shares are sampled according to the scheme $\mathsf{RMStochSS}_{2^m-1}$, while the rest of the shares are empty strings. To recover the secret of a set $I \subseteq \{1,\ldots,n\}$, we execute the recovery algorithm of $\mathsf{RMStochSS}_{2^m-1}$ on the shares of the parties in the set $I \cap \{1,\ldots,2^m-1\}$.*

Theorem 3.6. *For every constant $0 < p < 1/2$ there exists a constant $\delta > 0$ such that for all sufficiently large n, the scheme $\mathsf{RMStochSS}_n = (\mathsf{Share}_n, \mathsf{Recover}_n)$ is a linear n-party p-stochastic secret sharing scheme with error $\epsilon = \epsilon(n) = 2^{-\delta \cdot \sqrt{n}}$ and share size of 1 bit for the domain of secrets $\{0,1\}$. In addition, the scheme is multiplicative, and the circuit size of Share_n is at most $O(n \cdot \log(n))$.*

Proof. We first provide a proof for the special case where $n = n(m) = 2^m - 1$, and then explain how to extend the proof for every value of n.

Secret sharing. Recall that C_m is the code $\mathsf{RM}(\lfloor (m-1)/2 \rfloor, m)$. Observe that the dual code of C_m, denoted $C_m^{\perp}$, is the code $\mathsf{RM}(m - \lfloor (m-1)/2 \rfloor - 1, m)$, and that, by definition, both C_m and $C_m^{\perp}$ are subsets of the code $\mathsf{RM}(\lfloor m/2 \rfloor, m)$. Since the block-MAP decoder errs if and only if the set of erasures covers a codeword (see Remark 2.1), we conclude that C_m and $C_m^{\perp}$ also achieve capacity over the BEC channel. That is, for every constant $0 < p < 1/2$ there exists a constant

$\delta > 0$ such that the error probability of the block-MAP decoder of C_m and $C_m^\perp$ in $\mathsf{BEC}(p)$ is at most $\exp(-\delta \cdot 2^{m/2})$ for all sufficiently large m. Therefore, by Lemma 3.3 we are guaranteed that every constant $0 < p < 1/2$ there exists a constant $\delta > 0$ so that for all sufficiently large m the scheme $\mathsf{RMStochSS}_{n(m)}$ is p-stochastic secret sharing with error $2^{-\delta\sqrt{n}}$.

Multiplication. To see that the secret sharing is multiplicative, we note that the sharing vectors $(s_1, \ldots, s_n)$ and $(s'_1, \ldots, s'_n)$ of secrets s and s' correspond to the evaluations of an m-variate polynomial of degree less than $m/2$. Therefore, the vector $(s_1 \cdot s'_1, \ldots, s_n \cdot s'_n)$ corresponds to the evaluations of an m-variate polynomial of degree at most $m-1$. The dual code of $\mathsf{RM}(m-1, m)$ is $\mathsf{RM}(0, m)$, that contains the all-1 vector $(1, 1, \ldots, 1)$. In particular, every codeword $(a_0, a_1, \ldots, a_n)$ of $\mathsf{RM}(m-1, m)$ satisfies that $a_0 = a_1 + \ldots + a_n$. Therefore, it holds that $s \cdot s' = s_1 \cdot s'_1 + \ldots + s_n \cdot s'_n$. In other words, the vector $(s_1 \cdot s'_1, \ldots, s_n \cdot s'_n)$ is a (non-randomized) n-out-of-n secret sharing of $s \cdot s'$.

The sharing procedure. Observe that the sharing procedure of $\mathsf{RMStochSS}_{n(m)}$ is equivalent to picking a random polynomial $f(x_1, \ldots, x_m)$ of degree at most $\lfloor (m-1)/2 \rfloor$ whose free coefficient is s, and setting the shares to be the evaluations of the polynomial on all the points in $\mathbb{F}_2^m$ except for $(0, \ldots, 0)$.

To bound the circuit size required to compute the shares of the secret (measured by the number of wires), it is enough to bound the circuit size required to encode a codeword in $\mathsf{RM}(\lfloor (m-1)/2 \rfloor, m)$. Observe that every polynomial $f(x_1, \ldots, x_m)$ of degree $\lfloor (m-1)/2 \rfloor$ can be written as $f(x_1, \ldots, x_m) = g(x_1, \ldots, x_{m-1}) + x_m \cdot h(x_1, \ldots, x_{m-1})$, where g has degree $\lfloor (m-1)/2 \rfloor$ and h has degree $\lfloor (m-1)/2 \rfloor - 1$. Denote by $\mathsf{SIZE}(r, m)$ the circuit size of the encoder of $\mathsf{RM}(r, m)$, and observe that $\mathsf{SIZE}(\lfloor (m-1)/2 \rfloor, m) \leq \mathsf{SIZE}(\lfloor (m-1)/2 \rfloor, m-1) + \mathsf{SIZE}(\lfloor (m-1)/2 \rfloor - 1, m-1) + 2^m$. Therefore, one can verify that $\mathsf{SIZE}(\lfloor (m-1)/2 \rfloor, m) \leq O(2^m \cdot m) = O(n \cdot \log n)$.

Extending the construction for every n. It is not hard to verify that for $2^m - 1 < n < 2^{m+1} - 1$, the secret sharing scheme $\mathsf{RMStochSS}_n$ achieves the same security guarantees as $\mathsf{RMStochSS}_{2^m-1}$. That is, if $\mathsf{RMStochSS}_{2^m-1}$ is a $(2^m - 1)$-party p-stochastic secret sharing with error $\epsilon(2^m - 1)$ then $\mathsf{RMStochSS}_n$ is an n-party p-stochastic secret sharing with error $\epsilon(n) = \epsilon(2^m - 1)$, so the error of $\mathsf{RMStochSS}_n$ is the same as the error of $\mathsf{RMStochSS}_{2^m-1}$. This concludes the proof of the theorem. □

3.3 Stochastic Secret Sharing with Linear-Size Circuits

In this section we prove Theorem 1.2. Our construction is based on the linear uniform output family constructed by [DI14].

Definition 3.7 (Linear uniform output family [DI14])**.** *Let $\mathbb{F}_q$ be a finite field and let $k \leq n$ be positive integers. A (q, k, n)-*linear uniform output family *is a distribution $\mathcal{M}$ over $k \times n$ matrices over $\mathbb{F}_q$ that satisfies*

$$\Pr_{M \leftarrow \mathcal{M}}[\mathbf{x} \cdot M = \mathbf{y}] = q^{-n}$$

for every non-zero vector $\mathbf{x} \in \mathbb{F}_q^k$ and every vector $\mathbf{y} \in \mathbb{F}_q^n$.

We continue by showing that a linear uniform output family implies stochastic secret sharing. To do so, we first prove that a linear uniform output family implies good codes for erasure recovery over the QEC. Then, we use the main lemma (Lemma 3.3) to obtain stochastic secret sharing.

For a $k \times n$ matrix $M \in \mathbb{F}^{k \times n}$, we denote by $C(M)$ the linear code generated by M, and by $C(M)^\perp$ the linear code whose parity-check matrix is M. Observe that $C(M)^\perp$ is indeed the dual code of $C(M)$. The following theorem shows that with overwhelming probability over the choice of $M \leftarrow \mathcal{M}$, both $C(M)$ and $C(M)^\perp$ are good codes for erasure recovery over QEC.

Theorem 3.8. *Let $\mathbb{F}_q$ be a finite field, let k be a positive integer and let $n = 2k$. Let $\mathcal{M}$ be a (q, k, n)-linear uniform output family. Let $0 < \delta < 1/2$ and let $p = 1/2 - \delta$. Then, with probability at least $1 - 4 \cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of $M \leftarrow \mathcal{M}$ the following holds: (1) the probability that the block-MAP decoder of $C(M)$ fails in the q-ary erasure channel $\mathsf{QEC}(p)$ is at most $\sqrt{2} \cdot 2^{-\frac{\delta^2}{4}n}$, and (2) the the probability that the block-MAP decoder of $C(M)^\perp$ fails in the q-ary erasure channel $\mathsf{QEC}(p)$ is at most $\sqrt{2} \cdot 2^{-\frac{\delta^2}{4}n}$.*

Theorem 3.8 immediately follows from two more general lemmas: Lemma 3.13 that shows that $C(M)$ is a good erasure code, and is proved in Section 3.3.1; and Lemma 3.15 that shows that that $C(M)^\perp$ is a good erasure code, and is proved in Section 3.3.2. Using the Main Lemma 3.3 we immediately obtain the following corollary.

Corollary 3.9. *Let $\mathbb{F}_q$ be a finite field, let k be a positive integer and let $n+1 = 2k$. Let $\mathcal{M}$ be a $(q, k, n+1)$-linear uniform output family. Let $0 < \delta < 1/2$ and let $p = 1/2 - \delta$. With probability at least $1 - 4 \cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of $M \leftarrow \mathcal{M}$ the following holds. The scheme $\mathsf{StochSS}(C(M))$ is a linear n-party p-stochastic secret sharing scheme over $\mathbb{F}_q$ with error $\epsilon = \epsilon(n) = \frac{4}{p} \cdot 2^{-\frac{\delta^2}{4}n}$ for the domain of secrets $\mathbb{F}_q$.*

To obtain an n-party secret sharing scheme for an even n, we can just take the construction for $(n+1)$-party secret sharing scheme and ignore the last share, at the cost of increasing the error probability from $\epsilon(n+1)$ to $\epsilon(n+1)/p$.

Linear time stochastic secret sharing. It remains to prove that there exists a (q, k, n)-linear uniform output family $\mathcal{M}$ for which we can implement the sharing function of $\mathsf{StochSS}(C(M))$ by a linear-size circuit. For a $k \times n$ matrix M, let $A_M(\mathbf{x}) : \mathbb{F}_q^k \to \mathbb{F}_q^n$ be the linear function corresponding to left multiplication by M, i.e., $A_M(\mathbf{x}) = \mathbf{x} \cdot M$. The work of [DI14] constructs a linear uniform output family with *linear-size* circuits for A_M, as summarized in the following theorem.

Theorem 3.10 ([DI14])**.** *Let $\mathbb{F}_q$ be a finite field of size q, and let c be a positive integer. Then there exist (1) a family of distributions $(\mathcal{M}_k)_{k>1}$ such that $\mathcal{M}_k$ is a (q, k, ck)-linear uniform output family, and (2) a probabilistic polynomial-time algorithm Gen that takes as an input 1^k and samples an arithmetic circuit of size $O(k)$ that computes $A_M(\mathbf{x})$, where M is distributed according to $\mathcal{M}_k$.*

In particular, by taking $c = 2$, we get that $\mathcal{M}_k$ is a family of $(q, k, n+1 = 2k)$-linear uniform output family, with circuit of size $O(n)$ for A_M. As observed by [DI14], for every M in the support of $\mathcal{M}_k$, we can implement the sharing procedure of $\mathsf{StochSS}(C(M))$ by a linear-size circuit. Indeed, let us fix a global codeword $\mathbf{c}^* \in C(M)$, $\mathbf{c}^* = (c_0^*, c_1^* \ldots, c_n^*)$ with $c_0^* \neq 0$. To pick a random codeword whose 0th entry is s, we do as follows: (1) we pick a random vector $\mathbf{c}' \in C(M)$ by computing $\mathbf{c}' = A_M(\mathbf{x})$ for $\mathbf{x} \leftarrow \mathbb{F}_q^k$, (2) compute $\mathbf{c} = \mathbf{c}' + \alpha \cdot \mathbf{c}^*$ where $\alpha \in \mathbb{F}_q$ is defined to be

$\alpha = \frac{s-c'_0}{c^*_0}$. By Theorem 3.10 $A_M(\mathbf{x})$ has a linear-size circuit, and it is not hard to see that, since $\mathbf{c}^*$ is fixed, the second step can also be computed by a linear size circuit. We therefore obtain the following corollary.

Corollary 3.11. *Let $\mathbb{F}_q$ be a finite field. There exists a probabilistic polynomial time algorithm* Gen *that on input 1^n samples the description of an n-party linear secret sharing scheme $S_n = (\mathsf{Share}_n, \mathsf{Recover}_n)$ over $\mathbb{F}_q$, such that Share_n is represented by a circuit of size $O(n)$, and for every $0 < \delta < 1/2$ and $p = 1/2 - \delta$, with probability at least $1 - 4 \cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of S_n, the scheme S_n is p-stochastic secret sharing with error $\epsilon = \epsilon(n) = \frac{4}{p^2} \cdot 2^{-\frac{\delta^2}{4}n}$ for the domain of secrets $\mathbb{F}_q$.*

Theorem 1.2 now follows by taking the finite field to be $\mathbb{F}_2$.

Remark 3.12 (On 2-bits multiplicative secret sharing.)**.** *The above scheme is not multiplicative. However, using the technique of [CDM00, CCG+07], we can turn it into a* multiplicative *stochastic secret sharing scheme while preserving the exponentially-small error. This is done at the expense of increasing the share size to 2 bits, and the computation time to $O(n^2)$.*

The idea is to share a secret $s \in \{0,1\}$ by handing a pair of bits (s_i, s'_i) to the ith party where the vector $(s_1, \ldots, s_n)$ is sampled according to $\mathsf{StochSS}(C(M))$ and the vector $(s'_1, \ldots, s'_n)$ is sampled according to $\mathsf{StochSS}(C(M)^\perp)$. By Corollary 3.9 it is not hard to see that if M is sampled from a linear uniform output family, then with all but exponentially-small probability this scheme is a stochastic secret sharing scheme. In addition, the scheme is multiplicative, since for two secret $s, t \in \{0,1\}$ with shares $((s_1, s'_1), \ldots, (s_n, s'_n))$ and $((t_1, t'_1), \ldots, (t_n, t'_n))$, respectively, it holds that $s \cdot t = s_1 \cdot t'_1 + \ldots + s_n \cdot t'_n$. To see this, simply observe that $(s, s_1, \ldots, s_n) \in C(M)$ and $(t, t'_1, \ldots, t'_n) \in C(M)^\perp$ so $s \cdot t + s_1 \cdot t'_1 + \ldots + s_n \cdot t'_n = 0$.

3.3.1 $C(M)$ is a Good Erasure Code

In this section we prove the following lemma.

Lemma 3.13. *Let $\mathbb{F}_q$ be a finite field of size q, let $0 < p < 1$, let $\delta > 0$, and let $k \leq n$ be positive integers that satisfy $k/n = 1 - p - \delta$. Let $\mathcal{M}$ be a (q, k, n)-linear uniform output family. Then with probability at least $1 - \sqrt{2} \cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of $M \leftarrow \mathcal{M}$, the probability that the block-MAP decoder of $C(M)$ fails in the q-ary erasure channel $\mathsf{QEC}(p)$ is at most $\sqrt{2} \cdot 2^{-\frac{\delta^2}{4}n}$.*

To prove Lemma 3.13 we need the following claim.

Claim 3.14. *Let $\mathbb{F}_q$ be a finite field, and let $\mathcal{M}$ be a (q, k, n)-linear uniform output family. Let $I \subseteq [n]$ be a subset of size $\ell > k$, and denote by $M|_I$ the restriction of the matrix M to the columns in I. Then*

$$\Pr_{M \leftarrow \mathcal{M}}[\mathsf{rank}(M|_I) = k] \geq 1 - q^{-(\ell - k)}.$$

Proof. Observe that $\mathsf{rank}(M|_I) < k$ if and only if there exists a non-zero vector $\mathbf{x} \in \mathbb{F}_q^k$ such that $\mathbf{x} \cdot M|_I = 0$. Therefore,

$$\begin{aligned}
\Pr_{M \leftarrow \mathcal{M}}[\mathsf{rank}(M|_I) < k] &= \Pr_{M \leftarrow \mathcal{M}}[\exists \text{ non-zero } \mathbf{x} \in \mathbb{F}_q^k \text{ s.t. } \mathbf{x} \cdot M|_I = 0] \\
&\leq q^k \cdot q^{-\ell} \\
&= q^{-(\ell - k)},
\end{aligned}$$

where the inequality follows by union bound, and since $\mathcal{M}$ is a (q, k, n)-linear uniform output family. This completes the proof of the claim. □

We continue with the proof of Lemma 3.13.

Proof of Lemma 3.13. In the following we denote by $J \subseteq \{1,\ldots,n\}$ the random variable that corresponds to the set of coordinates erased by the $\mathsf{QEC}(p)$ channel, and by $I = \{1,\ldots,n\} \setminus J$ the set of non-erased coordinates. By Chernoff's bound, the probability that the size of J is more than $(p+\delta/2)n$ is at most $\exp(-2\cdot\delta^2 n/4) = \exp(-\delta^2 n/2)$. Let $M|_I$ be the matrix M when restricted to the columns of non-erased coordinates, and observe that the block-MAP decoder fails if and only if the k rows of $M|_I$ are linearly dependent. Now,

$$\begin{aligned}
\Pr_{\substack{M\leftarrow\mathcal{M}\\(J,I=[n]\setminus J)}}[\mathsf{rank}(M|_I)=k] &\geq \sum_{s=0}^{\lfloor(p+\delta/2)n\rfloor} \Pr_{\substack{M\leftarrow\mathcal{M}\\(J,I=[n]\setminus J)}}[\mathsf{rank}(M|_I)=k \mid |J|=s]\cdot\Pr[|J|=s] \\
&\geq (1-q^{-(n-(p+\delta/2)n)+k}) \sum_{s=0}^{\lfloor(p+\delta/2)n\rfloor} \Pr[|J|=s] \\
&= (1-q^{-(1-(p+\delta/2))n+(1-p-\delta)n})\cdot\Pr[|J|\leq\lfloor(p+\delta/2)n\rfloor] \\
&\geq (1-q^{-n\cdot\delta/2})(1-\exp(-\delta^2 n/2)) \\
&> 1-2\cdot 2^{-\frac{\delta^2}{2}\cdot n},
\end{aligned}$$

where we used Claim 3.14 in the second inequality, using the fact that $|I| = n - |J| \geq n - \lfloor(p+\delta/2)n\rfloor \geq (1-(p+\delta/2))n > (1-p-\delta)n = k$ for every J with $|J| \leq \lfloor(p+\delta/2)n\rfloor$. Then, by Fact 2.4, with probability at least $1-\sqrt{2}\cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of $M \leftarrow \mathcal{M}$ it holds that the probability that the block-MAP decoder of $C(M)$ fails is at most $\sqrt{2}\cdot 2^{-\frac{\delta^2}{4}n}$. This completes the proof of the lemma. □

3.3.2 $C(M)^\perp$ is a Good Erasure Code

In this section we prove the following lemma.

Lemma 3.15. *Let $\mathbb{F}_q$ be a finite field of size q, let $0<p<1$, let $\delta>0$, and let $k \leq n$ be positive integers that satisfy $(n-k)/n = 1-p-\delta$. Let $\mathcal{M}$ be a (q,k,n)-linear uniform output family. Then with probability at least $1-\sqrt{2}\cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of $M \leftarrow \mathcal{M}$, the probability that the block-MAP decoder of $C(M)^\perp$ fails in the q-ary erasure channel $\mathsf{QEC}(p)$ is at most $\sqrt{2}\cdot 2^{-\frac{\delta^2}{4}n}$.*

To prove Lemma 3.15 we need the following XOR-lemma, due to [CGH+85]. The following version is taken from [DI14, Lemma 2].

Lemma 3.16 (XOR-lemma)**.** *Let $\mathbf{x} = (x_1,\ldots,x_n)$ be a random variable, distributed over $\mathbb{F}_q^n$. Then $\mathbf{x}$ is uniformly distributed if and only if for every non-zero vector $(y_1,\ldots,y_n) \in \mathbb{F}_q^n$ it holds that the random variable $y_1\cdot x_1 + \ldots + y_n\cdot x_n$ is uniformly distributed over $\mathbb{F}_q$.*

The XOR-lemma implies that for every (q,k,n)-linear uniform output family $\mathcal{M}$ the following holds:

$$\Pr_{M\leftarrow\mathcal{M}}[M\cdot\mathbf{x}^T = \mathbf{z}^T] = q^{-k},$$

for every non-zero $\mathbf{x} \in \mathbb{F}_q^n$ and every $\mathbf{z} \in \mathbb{F}_q^k$. Indeed, for every non-zero vector $\mathbf{y} \in \mathbb{F}_q^k$ the random variable $\mathbf{y} \cdot M$ is uniformly distributed, since M is sampled from a (q, k, n)-linear uniform output family. Therefore, for every non-zero $\mathbf{x} \in \mathbb{F}_q^n$ and every non-zero vector $\mathbf{y} \in \mathbb{F}_q^k$ the random variable $\mathbf{y} \cdot M \cdot \mathbf{x}^T$ is uniformly distributed. Hence, by the XOR-lemma, for every non-zero vector $\mathbf{x} \in \mathbb{F}_q^n$, the random variable $M \cdot \mathbf{x}^T$ is uniformly distributed. We continue with the following claim, that is required for the proof of Lemma 3.15.

Claim 3.17. *Let $\mathbb{F}_q$ be a finite field, and let $\mathcal{M}$ ba a (q, k, n)-linear uniform output family. Let $J \subseteq [n]$ be a subset of size $\ell < k$, and denote by $M|_J$ the matrix M when restricted to the columns in J. Then*

$$\Pr_{M \leftarrow \mathcal{M}}[\mathsf{rank}(M|_J) = \ell] \geq 1 - q^{-(k-\ell)}.$$

Proof. Observe that $\mathsf{rank}(M|_J) < \ell$ if and only if there exists a non-zero vector $\mathbf{x} \in \mathbb{F}_q^n$ whose support is a subset of J such that $M \cdot \mathbf{x}^T = 0$. Therefore,

$$\begin{aligned}
\Pr_{M \leftarrow \mathcal{M}}[\mathsf{rank}(M|_J) < \ell] &= \Pr_{M \leftarrow \mathcal{M}}[\exists \text{ non-zero } \mathbf{x} \in \mathbb{F}_q^n \text{ with support in } J \text{ s.t. } M \cdot \mathbf{x}^T = 0] \\
&\leq q^{\ell} \cdot q^{-k} \\
&= q^{-(k-\ell)},
\end{aligned}$$

where the inequality follows by union bound, and since $M \cdot \mathbf{x}^T$ is uniformly distributed over $\mathbb{F}^k$ for $M \leftarrow \mathcal{M}$. This completes the proof of the claim. □

We continue with the proof of Lemma 3.15.

Proof of Lemma 3.15. In the following we denote by $J \subseteq \{1, \ldots, n\}$ the random variable that corresponds to the set of coordinates erased by the $\mathsf{QEC}(p)$ channel, and by $I = \{1, \ldots, n\} \setminus J$ the set of non-erased coordinates. By Chernoff's bound, the probability that the size of J is more than $(p + \delta/2)n$ is at most $\exp(-2 \cdot \delta^2 n/4) = \exp(-\delta^2 n/2)$.

Recall that the block-MAP decoder of $C(M)^{\perp}$ fails on J if and only if J covers a codeword in $C(M)^{\perp}$. Since M is the parity check matrix of $C(M)^{\perp}$, this happens if and only if the columns of $M|_J$ are linearly dependent. Now

$$\begin{aligned}
\Pr_{\substack{M \leftarrow \mathcal{M} \\ J}}[\mathsf{rank}(M|_J) = |J|] &\geq \sum_{s=0}^{\lfloor (p+\delta/2)n \rfloor} \Pr_{\substack{M \leftarrow \mathcal{M} \\ J}}[\mathsf{rank}(M|_J) = s \mid |J| = s] \cdot \Pr[|J| = s] \\
&\geq (1 - q^{-k+(p+\delta/2)n}) \cdot \sum_{s=0}^{\lfloor (p+\delta/2)n \rfloor} \Pr[|J| = s] \\
&= (1 - q^{-(p+\delta)n+(p+\delta/2)n}) \cdot \Pr[|J| < \lfloor (p + \delta/2)n \rfloor] \\
&\geq (1 - q^{-\delta \cdot n/2}) \cdot (1 - \exp(-\delta^2 n/2)) \\
&> 1 - 2 \cdot 2^{-\frac{\delta^2}{2} \cdot n}
\end{aligned}$$

where we used Claim 3.17 in the second inequality, using the fact that $|J| = \lfloor (p + \delta/2)n \rfloor < (p + \delta)n = k$. Then, by Fact 2.4, with probability at least $1 - \sqrt{2} \cdot 2^{-\frac{\delta^2}{4} n}$ over the choice of $M \leftarrow \mathcal{M}$ it holds that the probability that the block-MAP decoder of $C(M)^{\perp}$ fails is at most $\sqrt{2} \cdot 2^{-\frac{\delta^2}{4} n}$. This completes the proof of the lemma. □

4 Static Secret Sharing

In this section we show a general transformation from p-stochastic secret sharing to t-static secret sharing, and also prove a lower bound on 1-bit static secret sharing. We begin with a formal definition of t-static secret sharing.

Definition 4.1 (t-static secret sharing)**.** *Let $t \leq n$ be positive integers, let $\epsilon > 0$, and let $\mathcal{S}$ be a finite set of size at least 2. An n-party t-static secret sharing scheme with error ϵ for a domain of secrets $\mathcal{S}$ is a triple of algorithms* $(\mathsf{GenRand}, \mathsf{Share}, \mathsf{Recover})$ *such that*

- $\mathsf{GenRand}$ *is a randomized algorithm that outputs a string of public randomness R.*
- Share *is a randomized algorithm, that takes a secret $s \in \mathcal{S}$, the public randomness R, and private randomness r, and returns n shares $s_1, \ldots, s_n$.*
- $\mathsf{Recover}$ *is a deterministic algorithm, that takes a set $I \subseteq \{1, \ldots, n\}$, shares $(s_i)_{i \in I}$, and the public randomness R, and either returns some element $s' \in \mathcal{S}$ or a failure symbol $\perp$.*

The algorithms satisfy the following properties. For every set $\mathsf{Corrupt} \subseteq \{1, \ldots, n\}$ of size at most t, with probability at least $1 - \epsilon$ over the choice of R, the following properties hold:

- (Correctness) *For every secret $s \in \mathcal{S}$, and every string r of private randomness, it holds that $\mathsf{Recover}_R(\mathsf{Honest}, (s_i)_{i \in \mathsf{Honest}}) = s$, where $(s_1, \ldots, s_n) = \mathsf{Share}_R(s; r)$ and $\mathsf{Honest} = \{1, \ldots, n\} \setminus \mathsf{Corrupt}$.*
- (Privacy) *For every pair of secrets $s, s' \in \mathcal{S}$, it holds that the random variables $(s_i)_{i \in \mathsf{Corrupt}}$ have the same distribution as the random variables $(s'_i)_{i \in \mathsf{Corrupt}}$, where $(s_1, \ldots, s_n)$ are sampled from $\mathsf{Share}_R(s; r)$ and $(s'_1, \ldots, s'_n)$ are sampled from $\mathsf{Share}_R(s'; r)$, and r is uniformly distributed.*

A t-static secret sharing scheme is linear if for every choice of the public randomness R, the secret sharing scheme defined by $(\mathsf{Share}_R, \mathsf{Recover}_R)$ is linear. The definition of multiplicative secret sharing scheme extends in the same way as well.

4.1 From Stochastic Corruptions to Standard Corruptions

The transformation from p-stochastic secret sharing to t-static secret sharing is presented in the following construction.

Construction 4.2. *Let $(\mathsf{Share}', \mathsf{Recover}')$ be an n-party p-stochastic secret sharing with error ϵ. Consider the following secret sharing scheme* $(\mathsf{GenRand}, \mathsf{Share}, \mathsf{Recover})$ *with public randomness:*

- (Public randomness) *The algorithm* $\mathsf{GenRand}$ *samples a random permutation π of $\{1, \ldots, n\}$ and outputs π.*
- (Sharing) *The randomized algorithm* Share *takes as an input a secret s and the public randomness π, samples randomness r for Share', and computes $(s'_1, \ldots, s'_n) := \mathsf{Share}'(s; r)$. For every $i \in \{1, \ldots, n\}$ the algorithm sets $s_i := s'_{\pi(i)}$ and outputs $(s_1, \ldots, s_n)$.*
- (Recovery) *The algorithm* $\mathsf{Recover}$ *takes as an input a set $I \subseteq \{1, \ldots, n\}$, shares $(s_i)_{i \in I}$, and the public randomness π, sets $I' := \pi(I) = \{\pi(i) : i \in I\}$ and $s'_{\pi(i)} := s_i$ for every $i \in I$, and returns $\mathsf{Recover}'(I', (s'_i)_{i \in I'})$.*

Lemma 4.3. *Let n be a positive integer, let $\mathcal{S}$ be a finite set, let $0 < \epsilon' < 1$ and $0 < p < 1/2$, and let $S' = (\mathsf{Share}', \mathsf{Recover}')$ be an n-party p-stochastic secret sharing with error ϵ' for the domain of secret $\mathcal{S}$. Let $S = (\mathsf{GenRand}, \mathsf{Share}, \mathsf{Recover})$ be the scheme obtained by applying Construction 4.2 on S'. Then the scheme S is an n-party $\lfloor pn \rfloor$-static secret sharing with error $\epsilon \leq 2\epsilon'$ for the domain of secrets $\mathcal{S}$. In addition, if S' is linear (resp., multiplicative) then S is linear (resp., multiplicative).*

Proof. Fix any set $\mathsf{Corrupt} \subseteq \{1, \ldots, n\}$ of $t \leq \lfloor pn \rfloor$ corrupt parties, and let $\mathsf{Honest} = \{1, \ldots, n\} \setminus \mathsf{Corrupt}$ be the set of the honest parties. Let π be a random permutation, and observe that the random variable $\pi(\mathsf{Corrupt}) = \{\pi(i) : i \in \mathsf{Corrupt}\}$ is uniformly distributed over sets of size t of $\{1, \ldots, n\}$. Our goal is to prove that in the scheme S' both correctness and privacy hold for the set $\pi(\mathsf{Corrupt})$ of corrupt parties, with probability at least $1 - \epsilon$ over the choice of π.

Let $J \subseteq \{1, \ldots, n\}$ be the random variable that corresponds to the set of corrupt parties in the p-stochastic corruption model, and let BAD be the set of all subsets $L \subseteq \{1, \ldots, n\}$ such that if $J = L$ then either correctness or privacy (or both) in the scheme S' do not hold. For every integer $0 \leq m \leq n$, we denote by $J \mid_{|J|=m}$ the random variable J conditioned on the event that $|J| = m$, and we note that it is uniformly distributed over subsets of $\{1, \ldots, n\}$ of size m. Since $J \mid_{|J|=t}$ has the same distribution as $\pi(\mathsf{Corrupt})$, our goal is to prove that with probability at least $1 - \epsilon$ it holds that $J \mid_{|J|=t} \notin \mathsf{BAD}$. For this, we need the following claim.

Claim 4.4. *For every integers $0 \leq m < m' \leq n$ it holds that*

$$\Pr[J \in \mathsf{BAD} \mid |J| = m'] \geq \Pr[J \in \mathsf{BAD} \mid |J| = m].$$

Proof. It is enough to prove that $\Pr[J \in \mathsf{BAD} \mid |J| = m + 1] \geq \Pr[J \in \mathsf{BAD} \mid |J| = m]$, since the claim follows by induction. Now, $J \mid_{|J|=m+1}$ is uniform over subsets of $\{1, \ldots, n\}$ of size $m + 1$, and we think of picking such a set as a 2-step procedure: (1) pick a random subsets $J' \subseteq \{1, \ldots, n\}$ of size m, and (2) pick an additional integer from $\{1, \ldots, n\} \setminus J'$ at random. Therefore, $\Pr[J \in \mathsf{BAD} \mid |J| = m + 1] \geq \Pr[J' \in \mathsf{BAD} \mid |J| = m + 1] = \Pr[J \in \mathsf{BAD} \mid |J| = m]$. This completes the proof of the claim. □

We observe that $|J|$ is at least t with probability at least $1/2$. Indeed, it is known that any median of the Binomial distribution with n trials and success probability p in each trial lies between $\lfloor np \rfloor$ to $\lceil np \rceil$ (see [KB80]). Therefore,

$$\begin{aligned}
\epsilon' &\geq \Pr[J \in \mathsf{BAD}] \\
&= \sum_{m=0}^{n} \Pr[J \in \mathsf{BAD} \mid |J| = m] \cdot \Pr[|J| = m] \\
&\geq \sum_{m=t}^{n} \Pr[J \in \mathsf{BAD} \mid |J| = m] \cdot \Pr[|J| = m] \\
&\geq \Pr[J \in \mathsf{BAD} \mid |J| = t] \cdot \sum_{m=t}^{n} \Pr[|J| = m] \\
&= \Pr[J \in \mathsf{BAD} \mid |J| = t] \cdot \Pr[|J| \geq t] \\
&\geq \Pr[J \in \mathsf{BAD} \mid |J| = t] \cdot \frac{1}{2},
\end{aligned}$$

where in the third inequality we used Claim 4.4. We conclude that $\Pr[J \in \mathsf{BAD} \mid |J| = t] \leq 2\epsilon'$. Finally, it is not hard to see that if S' is linear (resp., multiplicative), then so is S. This completes the proof of the lemma. □

t-static secret sharing from Reed-Muller code. For every positive integer n, let $\mathsf{RMStatSS}_n$ be the static secret sharing scheme obtained by applying Construction 4.2 on $\mathsf{RMStochSS}_n$ from Theorem 3.6. Then Lemma 4.3 immediately implies the following theorem, that corresponds to Corollary 1.3.

Theorem 4.5. *For every constant $0 < p < 1/2$ there exists a constant δ such that for all sufficiently large n, the scheme $\mathsf{RMStatSS}_n = (\mathsf{GenRand}_n, \mathsf{Share}_n, \mathsf{Recover}_n)$ is a linear n-party $\lfloor pn \rfloor$-static secret sharing scheme with error $\epsilon = \epsilon(n) = 2^{-\delta \cdot \sqrt{n}}$, and share size of 1 bit for the domain of secrets $\{0,1\}$. In addition, the scheme is multiplicative, and can be computed in time $O(n \cdot \log(n))$ in the RAM model.*

t-static secret sharing from linear uniform output family. Similarly, if we apply Lemma 4.3 on the stochastic secret sharing scheme that is based on linear uniform output family (Corollary 3.11) then we obtain the following theorem, that corresponds to Corollary 1.4 when taking the field to be $\mathbb{F}_2$.

Theorem 4.6. *Let $\mathbb{F}_q$ be a finite field. There exists a probabilistic polynomial time algorithm Gen that on input 1^n samples the description of an n-party secret sharing scheme $S_n = (\mathsf{GenRand}_n, \mathsf{Share}_n, \mathsf{Recover}_n)$ over $\mathbb{F}_q$, such that for every $0 < \delta < 1/2$, with probability at least $1 - 4 \cdot 2^{-\frac{\delta^2}{4}n}$ over the choice of S_n, the scheme S_n is $\lfloor (1/2 - \delta)n \rfloor$-static linear secret sharing scheme with error $\epsilon = \epsilon(n) = \frac{8}{(1/2-\delta)^2} \cdot 2^{-\frac{\delta^2}{4}n}$ for the domain of secrets $\mathbb{F}_q$. Algorithm $\mathsf{GenRand}_n$ can be implemented in time $O(n \cdot \log(n))$ in the RAM model, and outputs a public string of length $O(n \cdot \log(n))$. In addition, Share_n requires a preprocessing time of $O(n \cdot \log(n))$, and after that sharing a secret requires $O(n)$ time in the RAM model.*

4.2 Lower Bound on Static Secret Sharing

In this section we prove the following lower bound on static secret sharing.

Theorem 4.7. *For every positive integer $n \geq 5$, and every $0 < c < (n-4)/6$ there is no 1-bit n-party $(\frac{1}{2} - \delta)n$-static secret sharing scheme with error ϵ for the domain of secrets $\{0,1\}$, where $\delta = \frac{c}{n}$ and $\epsilon = 2^{-6c-6}$.*

In particular, when $\delta = O(1/n)$ then $\epsilon = \Omega(1)$, and when $\delta = O(\log(n)/n)$ then $\epsilon \geq 1/\mathrm{poly}(n)$. We mention that the theorem can be extended to stochastic secret sharing as well (with slight degradation of the parameters) using Lemma 4.3. Proving Theorem 4.7 is done in two steps: First, we prove that any $(\frac{1}{2} - \delta)n$-static secret sharing scheme with sufficiently low error ϵ can be translated into a (standard) k-party $(\frac{k}{2} - \delta n - 1, \frac{k}{2} + \delta n + 1)$-ramp secret sharing; Then, we use the lower bounds of [BGK20] on ramp secret sharing to get a contradiction.

For the first part, we have the following lemma. (In this section, all logarithms are base 2.)

Lemma 4.8. *Let $\mathcal{S}$ be a finite set, let $n \geq 5$ be a positive integer, let $0 < \delta < \frac{1}{2} - \frac{2}{n}$ and $0 \leq \epsilon < 2^{-2\delta n - 5}$, and let $S = (\mathsf{GenRand}, \mathsf{Share}, \mathsf{Recover})$ be an n-party $(\frac{1}{2} - \delta)n$-static secret sharing scheme with error ϵ for the domain of secrets $\mathcal{S}$. Assume that the share size of S is ℓ bits. Then for $k = \min\{n, \lfloor \log(1/\epsilon) \rfloor\}$ there exists a k-party (t_p, t_c)-ramp secret sharing scheme for the domain of secrets $\mathcal{S}$ with share size of ℓ bits, where $t_p = \frac{k}{2} - \delta n - 1$ and $t_c = \frac{k}{2} + \delta n + 1$.*

Let us first observe that the parameters make sense. First, we note that $k > 2\delta n + 4$. Indeed, if $k = n$ then this follows from the requirement $\delta < \frac{1}{2} - \frac{2}{n}$; Otherwise, if $k = \lfloor \log(1/\epsilon) \rfloor$ then this follows from the requirement $\epsilon < 2^{-2\delta n - 5}$. As a consequence we obtain that the number of parties k in the new scheme is at least 5, that $t_p = \frac{k}{2} - \delta n - 1 > 1$ and that $t_c = \frac{k}{2} + \delta n + 1 < k - 1$. We continue with the proof of the lemma.

Proof of Lemma 4.8. We split into cases.

Case 1: $\epsilon < 2^{-n}$. In this case $n \leq \lfloor \log(1/\epsilon) \rfloor$ so we set $k = n$. It suffices to prove that there exists a public random string R^* such that if we execute the scheme S with the public random string fixed to R^*, hereafter denoted by S_{R^*}, then (1) correctness holds with respect to every set of size at least $(\frac{1}{2} + \delta)n$, and (2) privacy holds with respect to every set of size at most $(\frac{1}{2} - \delta)n$. The proof uses the probabilistic method and a simple union-bound. Specifically, fix a subset $Q \subseteq \{1, \ldots, n\}$ of size at most $(\frac{1}{2} - \delta)n$, and let us say that Q *fails* with respect to public randomness R if either (1) correctness fails with respect to the set of honest parties $\{1, \ldots, n\} \setminus Q$, or (2) privacy fails with respect to the set of corrupt parties Q. Then, by definition, Q fails for a randomly chosen R with probability of at most 2ϵ. By a union-bound over all 2^{n-1} subsets $Q \subseteq \{1, \ldots, n\}$ of size at most $(\frac{1}{2} - \delta)n$, we conclude that

$$\Pr_R[\exists Q \subseteq \{1, \ldots, n\},\ |Q| \leq (\frac{1}{2} - \delta)n,\ \text{s.t. } Q \text{ fails with respect to } R] \leq 2^{n-1} \cdot 2\epsilon < 1,$$

as required.

Case 2: $\epsilon \geq 2^{-n}$. In the second case we assume that $2^{-n} \leq \epsilon < 2^{-2\delta n - 5}$, so $\lfloor \log(1/\epsilon) \rfloor \leq n$ and we set $k = \lfloor \log(1/\epsilon) \rfloor$. We define three sets of parties, A, B and C, where $A = \{1, \ldots, k\}$, $B = \{k+1, \ldots, k + \lfloor (n-k)/2 \rfloor\}$ (so $|B| = \lfloor (n-k)/2 \rfloor$), and $C = \{1, \ldots, n\} \setminus (A \cup B)$. Observe that B and C might be empty. Looking forward, A will be the set of parties in the new ramp secret sharing scheme, while the shares of parties in B will be public and known to all the parties in A. Finally, we will ignore the shares of parties in C. We begin with the following formal claim.

Claim 4.9. *There is a choice of R^* for which the following holds:*

- *Every subset A' of size at most $(k/2 - \delta n - 1)$ of A has no information about the secret, even given the shares of parties in B. (That is, privacy holds with respect to the set $A' \cup B$.)*
- *Every subset A' of size at least $(k/2 + \delta n + 1)$ of A can recover the secret given the shares of parties in B. (That is, correctness holds with respect to the set $A' \cup B$.)*

Proof. For a subset $Q \subseteq A$ of size at most $\frac{k}{2} - \delta n - 1$, we say that Q *fails* with respect to the choice of the public randomness R generated by $\mathsf{GenRand}$ if either (1) correctness fails with respect to the set of honest parties $(A \setminus Q) \cup B$, or (2) privacy fails with respect to the set of corrupt parties $(Q \cup B)$. Since $|(A \setminus Q) \cup B| \geq (\frac{k}{2} + \delta n + 1) + \lfloor (n-k)/2 \rfloor \geq (1/2 + \delta)n$, then by the correctness of the scheme S we get that the correctness fails with probability at most ϵ over the choice of R. Similarly, since $|Q \cup B| \leq (\frac{k}{2} - \delta n - 1) + \lfloor (n-k)/2 \rfloor \leq (1/2 - \delta)n$, then by the privacy of the scheme S we get that the privacy fails with probability at most ϵ over the choice of R. Since the number of subsets $Q \subseteq A$ of size at most $(\frac{k}{2} - \delta n - 1)$ is less than 2^{k-1}, we conclude that

$$\Pr_R[\exists Q \subseteq A,\ |Q| \leq (\frac{k}{2} - \delta n - 1),\ \text{s.t. } Q \text{ fails with respect to } R] < 2^{k-1} \cdot 2\epsilon \leq 1.$$

Therefore, there exists public randomness R^* such that every subset $Q \subseteq A$ of size at most $(\frac{k}{2} - \delta n - 1)$ does not fail with respect to R^*. The claim follows. □

We continue with the definition of the ramp secret sharing scheme. Let $s^* \in \mathcal{S}$ be some secret, and let $(s_1^*, \ldots, s_n^*) = \mathsf{Share}_{R^*}(s^*; r^*)$ be the shares of s^* generated by Share when the public random string is fixed to R^* and the internal randomness is fixed to some string r^* (e.g., the all-zero string).

Consider the following k-party secret sharing scheme $S' = (\mathsf{Share}', \mathsf{Recover}')$:

- (Share'): To share a secret $s \in \mathcal{S}$, sample shares $(s_1, \ldots, s_n)$ according to $\mathsf{Share}_{R^*}(s; r)$ conditioned on $s_i = s_i^*$ for every $i \in B$. Output $(s_1, \ldots, s_k)$. (Recall that $A = \{1, \ldots, k\}$.)
- ($\mathsf{Recover}'$): For a set $I \subseteq \{1, \ldots, k\}$, given the shares $(s_i)_{i \in I}$ the recovery algorithm outputs $\mathsf{Recover}_{R^*}(I \cup B, (s_i)_{i \in I} \cup (s_i^*)_{i \in B})$.

We continue by proving that S' is a $(\frac{k}{2} - \delta n - 1, \frac{k}{2} + \delta n + 1)$-ramp secret sharing scheme. First, we observe that the sharing algorithm Share' is well defined. Indeed, by construction, we know that privacy in S holds with respect to the set B when the public string is R^*, and therefore for every secret $s \in \mathcal{S}$ there are shares $(s_1, \ldots, s_n)$ such that $(s_i)_{i \in B} = (s_i^*)_{i \in B}$. We conclude that for every secret $s \in \mathcal{S}$, the sharing algorithm Share' can sample shares $(s_1, \ldots, s_n) = \mathsf{Share}_{R^*}(s; r)$ conditioned on $s_i = s_i^*$ for all $i \in B$.

To see that correctness holds in S', fix any secret $s \in \mathcal{S}$, any set $I \subseteq \{1, \ldots, k\}$ of size at least $(\frac{k}{2} + \delta n + 1)$ and any shares $(s_i)_{i \in I}$ of s that were generated by Share'. Since the complement $Q = A \setminus I$ is a set of size at most $\frac{k}{2} - \delta n - 1$, then by construction correctness in S holds with respect to the set $I \cup B$ when the public string is R^*, and therefore $\mathsf{Recover}_{R^*}(I \cup B, (s_i)_{i \in I} \cup (s_i^*)_{i \in B}) = s$.

Finally, to see that privacy holds, fix any pair of secrets $s, s' \in \mathcal{S}$ and any set $Q \subseteq \{1, \ldots, k\}$ of size at most $(\frac{k}{2} - \delta n - 1)$. By construction, privacy in S holds with respect to the set $Q \cup B$ when the public string is R^*. That is, the shares $(s_i)_{i \in Q \cup B}$ have the same distribution as $(s_i')_{i \in Q \cup B}$ where $(s_1, \ldots, s_n) = \mathsf{Share}_{R^*}(s; r)$ and $(s_1', \ldots, s_n') = \mathsf{Share}_{R^*}(s'; r)$ and r is uniformly distributed. In particular, the shares $(s_i)_{i \in Q \cup B}$ have the same distribution as $(s_i')_{i \in Q \cup B}$ even conditioned on the event that the shares of parties in B are $(s_i^*)_{i \in B}$. Therefore the shares $(s_i)_{i \in Q}$ that are generated by Share' when the secret is s have the same distribution as the shares $(s_i')_{i \in Q}$ that are generated by Share' when the secret is s'. This concludes the proof of the lemma. □

We continue with the proof of Theorem 4.7.

Proof of Theorem 4.7. Fix $n \geq 5$ and $0 < c < (n-4)/6$. Assume towards contradiction that there exists a 1-bit n-party $(1/2 - \delta)$-static secret sharing scheme S with error ϵ for the domain of secrets $\{0, 1\}$, where $\delta = \frac{c}{n}$ and $\epsilon = 2^{-6c-6}$. Observe that $0 < \delta < \frac{1}{2} - \frac{2}{n}$ and $0 \leq \epsilon < 2^{-2\delta n - 5}$. Therefore, by Lemma 4.8, there exists a 1-bit k-party (t_p, t_c)-ramp secret sharing scheme S' for the domain of secrets $\{0, 1\}$, where $k = \min\{n, \lfloor \log(1/\epsilon) \rfloor\}$, $t_p = \frac{k}{2} - \delta n - 1$ and $t_c = \frac{k}{2} + \delta n + 1$. We split into cases.

- First, we assume that $n \leq \lfloor \log(1/\epsilon) \rfloor$, so $k = n$, $t_p = (\frac{1}{2} - \delta)n - 1$ and $t_c = (\frac{1}{2} + \delta n) + 1$. Therefore, the lower bound of [BGK20] implies that the share size of S' is at least $\log((n + t_c - t_p + 2)/(2(t_c - t_p))) = \log((n + 2c + 4)/(4c + 4)) > 1$ bits, where we used the fact that $c < (n-4)/6$. This contradicts the 1-bit share size of S'.

- Next, we assume that $n > \lfloor \log(1/\epsilon) \rfloor$ so $k = \lfloor \log(1/\epsilon) \rfloor$. A straightforward calculation shows that $6c+5 \leq k$, so $t_p \geq (2k-1)/6$ and $t_c \leq (4k+1)/6$. Therefore, the lower bound of [BGK20] implies that the share size of S' is at least $\log((k + t_c - t_p + 2)/(2(t_c - t_p))) > 1$ bits. This contradicts the 1-bit share size of S'.

This concludes the proof of the theorem. □

5 MPC in the Real-Time Computation Model

In this section we present our results regarding MPC in the real-time computation model. All our results are proved in the framework of universal composability (UC), the reader is referred to [Can01] for more details. Throughout, we will be interested in security against a passive adversary that corrupts a minority of the parties. We consider a single client and n servers and formalize the requirements from the real-time computation model via the following reactive functionality.

Functionality $\mathcal{F}_{\text{RT-MPC}}$

Public parameters: Positive integer m denoting the number of inputs.

Server's Inputs: The ith server inputs m bits $x_{1,i}, \ldots, x_{m,i} \in \mathbb{F}_2$.

Initialize registers: The functionality initializes $R = m$ registers, denoted $r_1, \ldots, r_R$. For $j = 1, \ldots, m$ the value of r_j is initialized to $x_j := x_{j,1} + \ldots + x_{j,n}$,.

Interaction with Client: Upon receiving an instruction inst from the client, the functionality returns inst to all the parties (the servers and the client) and acts as follows (all arithmetic operations are performed over $\mathbb{F}_2$).

- If inst $= (\mathsf{add}, i, j, k)$ for $i, j, k \in \{1, \ldots, R\}$, then the functionality sets the value of r_k to be the sum of r_i and r_j, i.e., $r_k \leftarrow r_i + r_j$.
- If inst $= (\mathsf{add1}, k)$ for $k \in \{1, \ldots, R\}$, then the functionality sets the value of r_k to be the value of r_k plus 1, i.e., $r_k \leftarrow r_k + 1$.
- If inst $= (\mathsf{multiply}, i, j, k)$ for $i, j, k \in \{1, \ldots, R\}$, then the functionality sets the value of r_k to be the multiplication of the value of r_i and the value of r_j, i.e., $r_k \leftarrow r_i \cdot r_j$.
- If inst $=$ increase then the functionality adds an additional register, initialized to 0, and increases the counter by 1, i.e., $R \leftarrow R + 1$.
- If inst $= (\mathsf{output}, i)$ for $i \in \{1, \ldots, R\}$ then the functionality returns the value of r_i to the client.

Figure 1: Functionality $\mathcal{F}_{\text{RT-MPC}}$

Remark 5.1 (On the choice of n-out-of-n secret sharing)**.** *Observe that the inputs of the servers form an n-out-of-n secret sharing of the bits $x_1, \ldots, x_m$. We mention that the choice of n-out-of-n secret sharing is arbitrary, and is done without loss of generality, since in the protocol we can always translate the n-out-of-n secret sharing of $x_1, \ldots, x_m$ into another secret sharing in time and communication that depend only on m and n, and are independent of the number of instructions B.*

The communication complexity. In the real-time model, the communication complexity of a protocol includes all the bits that the servers send throughout the execution of the protocol. We

emphasize that we *do not* count the instruction messages from the client to the server in the communication complexity (but we do count the messages from the servers to the client). Indeed, in many natural examples the instructions can either be deduced by the servers based on some external source (like in the example of secure rank computation in Section 1.3.1), or many instructions can be given in a succinct way. Finally, if the instructions cannot be represented in a succinct way (say, they are chosen at random), then *every protocol* has to communicate them to the servers.

5.1 Our Protocol

In this section we realize the functionality $\mathcal{F}_{\mathsf{RT\text{-}MPC}}$. In Section 5.1.1 we present an efficient algorithm for randomness extraction, and use it in Section 5.1.2 to construct our protocol. Finally, in Section 5.1.3 we discuss several variants of the basic protocol.

Settings and notations. Let $0 < \epsilon < 1/2$, and consider n servers $P_1, \ldots, P_n$ and a single client. A static, passive adversary can corrupt the client and up to t of the servers, where $t = (\frac{1}{2} - \epsilon)n$. We denote the set of corrupt servers by $\mathsf{Corrupt}$, and the set of honest servers by $\mathsf{Honest} = \{1, \ldots, n\} \setminus \mathsf{Corrupt}$. We assume that the parties have access to secure point-to-point channels, as well as to a broadcast channel.[8]

For simplicity we assume that n is of the form $n = 2^\alpha - 1$ for some integer α, and we later explain how to extend the protocol for every value of n. We let $\mathsf{RMStatSS} = (\mathsf{GenRand}^{\mathsf{RM}}, \mathsf{Share}^{\mathsf{RM}}, \mathsf{Recover}^{\mathsf{RM}})$ be the linear n-party t-static secret sharing scheme over $\mathbb{F}_2$ that is based on Reed-Muller codes (see Theorem 4.5). Since $n = 2^\alpha - 1$ then the share of each party is a single bit (i.e., none of the shares are empty strings, see Construction 3.5). We let $S^{\mathsf{n\text{-}n}} = (\mathsf{Share}^{\mathsf{n\text{-}n}}, \mathsf{Recover}^{\mathsf{n\text{-}n}})$ denote the standard n-out-of-n secret sharing scheme over $\mathbb{F}_2$. For a secret s we denote by $[s]^{\mathsf{RM}}_\pi$ the secret sharing of s using $\mathsf{RMStatSS}$ and public permutation π, where we assume that π is chosen once and for all. We also denote by $[s]^{\mathsf{n\text{-}n}}$ the secret sharing of s using $S^{\mathsf{n\text{-}n}}$.

We note that both $\mathsf{RMStatSS}$ and $S^{\mathsf{n\text{-}n}}$ are *linear* secret sharing schemes. Therefore, if two secrets s and s' are shared among the servers via $[s]$ and $[s']$, then a secret sharing of $s + s'$ can be obtained by locally summing-up the shares of s and s', which we denote by $[s+s'] = [s]+[s']$. This definition is extended naturally to a general linear function that is defined by a $k \times n$ matrix M, where we denote by $([s'_1], \ldots, [s'_k])^T = M \cdot ([s_1], \ldots, [s_n])^T$ the application of the linear function M on the vector of shares $([s_1], \ldots, [s_n])$ that results in a new vector of shares $([s'_1], \ldots, [s'_k])$.

5.1.1 Efficient Randomness Extraction with Passive Security

In this section we present an efficient protocol that allows the servers to efficiently generate many random secret-shared values. For this, we let every P_i share a random value s_i, and our goal is to combine the shares of $[s_1], \ldots, [s_n]$ into secret sharing of $\lfloor n/2 \rfloor$ random values that are not known to the adversary. Note that this task is not trivial, because the corrupt servers shared up to t of the original values.

[8] For a passive adversary, a broadcast channel can be emulated by simply sending the same message over the private channels. However, if the parties have access to a broadcast channel, we can further reduce the exact communication complexity, and therefore we analyse both cases. We emphasize that the computation and communication overhead, as well as the asymptotic complexity, remain the same with or without a broadcast channel.

To do so, we first present a simple probabilistic construction of a linear function that extracts $\lfloor n/2 \rfloor$ random bits out of a string of n bits that contains at least $n - t$ random bits in unknown *random* coordinates. This can be seen as a variant of the bit-extraction problem of [CGH+85]. Our main technical tool will be a $(2, \lfloor n/2 \rfloor, n)$-linear uniform output family (see Definition 3.7).

Randomness extractor. Let $\mathcal{M}$ be a $(2, \lfloor n/2 \rfloor, n)$-linear uniform output family. Let $J \subseteq \{1, \ldots, n\}$ be a random set of size t, and let $I = \{1, \ldots, n\} \setminus J$ be its complement of size $n - t > \lfloor n/2 \rfloor$. Let $\mathbf{x} \in \mathbb{F}_2^n$ be a random string, and denote the restriction of $\mathbf{x}$ to coordinates in J by $\mathbf{x}|_J$.

Our goal is to construct an $\lfloor n/2 \rfloor \times n$ matrix M over $\mathbb{F}_2$, such that with high probability over the choice of J, the random variable $(\mathbf{x}|_J, M \cdot \mathbf{x}^T)$ is uniformly distributed over $\mathbb{F}_2^{t+\lfloor n/2 \rfloor}$. Indeed, this would imply that with high probability over the choice of J, and for every fixing of the t entries $\mathbf{x}|_J$, the random variable $M \cdot \mathbf{x}^T$, which we think of as the extracted randomness of length $\lfloor n/2 \rfloor$, is uniformly distributed.

We continue with a probabilistic construction of such a matrix M, using the linear uniform output family $\mathcal{M}$. We say that a matrix M is "good", if the probability over the choice of J and $I = \{1, \ldots, n\} \setminus J$ that the random variable $(\mathbf{x}|_J, M \cdot \mathbf{x}^T)$ is uniformly distributed is at least $1 - 2^{-\epsilon \cdot n/2}$. We prove that picking a matrix according to $\mathcal{M}$ results in a good matrix with overwhelming probability. We begin with the following simple claim, where we denote by $M|_I$ the restriction of M to the columns in I.

Claim 5.2. *For every choice of M, J and $I = \{1, \ldots, n\} \setminus J$ such that the $\lfloor n/2 \rfloor \times (n-t)$ submatrix $M|_I$ has full rank, it holds that the random variable $(\mathbf{x}|_J, M \cdot \mathbf{x}^T)$ is uniformly distributed over $\mathbb{F}_2^{t+\lfloor n/2 \rfloor}$.*

Proof. By definition the random variable $\mathbf{x}|_J$ is uniformly distributed over $\mathbb{F}_2^t$. Fix this random variable. To see that $M \cdot \mathbf{x}^T$ is uniformly distributed even conditioned on $\mathbf{x}|_J$, we observe that $M \cdot \mathbf{x}^T = M|_J \cdot (\mathbf{x}|_J)^T + M|_I \cdot (\mathbf{x}|_I)^T$, where $M|_J \cdot (\mathbf{x}|_J)^T$ is some fixed vector. By assumption, $M|_I$ has full rank, i.e., $\mathsf{rank}(M|_I) = \lfloor n/2 \rfloor$. Therefore, the columns of $M|_I$ span $\mathbb{F}_2^{\lfloor n/2 \rfloor}$, and therefore the random variable $M|_I \cdot (\mathbf{x}|_I)^T$, which is just a random linear combination of the columns of $M|_I$, is uniformly distributed over $\mathbb{F}^{\lfloor n/2 \rfloor}$. This concludes the proof of the claim. □

The following lemma shows that a random choice of $M \leftarrow \mathcal{M}$ results in a good matrix with high probability.

Lemma 5.3. *The matrix M is good with probability at least $1 - 2^{-\epsilon \cdot n/2}$ over the choice $M \leftarrow \mathcal{M}$.*

Proof. By Claim 5.2 we notice that M is good if $\Pr_{(J, I=[n]\setminus J)}[\mathsf{rank}(M|_I) = \lfloor n/2 \rfloor] \geq 1 - 2^{-\epsilon \cdot n/2}$. From Claim 3.14 we know that for every choice of J and $I = \{1, \ldots, n\} \setminus J$, the matrix $M|_I$ has full rank with probability at least $1 - 2^{-\epsilon \cdot n}$, i.e., $\Pr_{M \leftarrow \mathcal{M}}[\mathsf{rank}(M|_I) = \lfloor n/2 \rfloor] \geq 1 - 2^{-\epsilon \cdot n}$. Therefore, by Fact 2.4, the matrix M is good with probability at least $1 - 2^{-\epsilon \cdot n/2}$, which completes the proof of the lemma. □

Our protocol. We continue with an efficient protocol for generating random secret-shared bits. Formally, we consider an *arbitrary* linear secret sharing scheme S over $\mathbb{F}_2$, denote the shares of a secret s by $[s]$, and assume that S provides privacy against the set of corrupt parties.[9] Our goal is

[9] For now, it will be convenient to think of S as either $S^{\mathsf{n\text{-}n}}$ or $\mathsf{RMStatSS}$. To simplify the presentation, when $S = \mathsf{RMStatSS}$, let us assume that the public randomness for $\mathsf{RMStatSS}$ was already generated, and condition on the event that $\mathsf{RMStatSS}$ provides correctness and privacy against the set of corrupt parties.

to implement the functionality $\mathcal{F}_{\mathsf{shareRandomValues}}$ that samples random bits $s'_1, \ldots, s'_{\lfloor n/2 \rfloor}$ and shares them among the servers via S.

The protocol has public parameters: a matrix M from a $(2, \lfloor n/2 \rfloor, n)$-linear uniform output family $\mathcal{M}$, and a permutation π of $\{1, \ldots, n\}$. We assume that (M, π) are picked once and for all, and can be used in many executions of the protocol. The protocol is standard: First, we let each server P_i share a random bit s_i to the other servers; then, we simply let the servers locally compute $\lfloor n/2 \rfloor$ secret-shared random values defined by the linear operation $([s'_1], \ldots, [s'_{\lfloor n/2 \rfloor}])^T = M \cdot ([s_{\pi(1)}], \ldots, [s_{\pi(n)}])^T$. Since the secret sharing scheme is linear, this computation is done locally by the servers, and requires no communication. Full details of the protocol are given in Figure 2.

Protocol shareRandomValues

Public parameters: A linear secret sharing scheme S over $\mathbb{F}_2$. An $\lfloor n/2 \rfloor \times n$ matrix $M \in \mathbb{F}^{\lfloor n/2 \rfloor \times n}$, and a permutation π of $\{1, \ldots, n\}$.

The protocol: The servers do as follows.

1. Each server P_i picks a random bit s_i and shares it among the servers via $[s_i]$.
2. The servers locally compute the linear function
$$([s'_1], \ldots, [s'_{\lfloor n/2 \rfloor}])^T = M \cdot ([s_{\pi(1)}], \ldots, [s_{\pi(n)}])^T.$$
3. The servers output their shares in $[s'_1], \ldots, [s'_{\lfloor n/2 \rfloor}]$.

Figure 2: Protocol shareRandomValues

Claim 5.4. *Assume that M is good, and that π is chosen at random at the beginning of the execution, and is independent of the set of corrupt parties. Then Protocol* shareRandomValues *is a statistically-secure UC-secure implementation of $\mathcal{F}_{\mathsf{shareRandomValues}}$ with error $1 - 2^{-\epsilon \cdot n/2}$, against an adversary that passively corrupts at most t of the parties.*

Proof sketch. First, we observe that the protocol merely consists of secure computation of the shares of $(s'_1, \ldots, s'_{\lfloor n/2 \rfloor})^T = M \cdot (s_{\pi(1)}, \ldots, s_{\pi(n)})^T$. Therefore, it is enough to prove that $s'_1, \ldots, s'_{\lfloor n/2 \rfloor}$ are uniformly distributed with overwhelming probability, even conditioned on the values of $(s_i)_{i \in \mathsf{Corrupt}}$ that are picked by the corrupt servers.

Assume that the matrix M is good. Observe that $\pi(\mathsf{Corrupt}) = \{\pi(i) : i \in \mathsf{Corrupt}\}$ is a uniformly distributed set of size at most t, and therefore with probability at least $1 - 2^{-\epsilon \cdot n/2}$ over the choice of π, the random variable $((s_i)_{i \in \mathsf{Corrupt}}, s'_1, \ldots, s'_{\lfloor n/2 \rfloor})$ is uniformly distributed. In particular, for every choice of $(s_i)_{i \in \mathsf{Corrupt}}$, the random variables $s'_1, \ldots, s'_{\lfloor n/2 \rfloor}$ are uniformly distributed, as required. This concludes the proof of the claim. □

Remark 5.5 (Generating shares of random bits using two secret sharing schemes.)**.** *In our final protocol, we would like to generate shares of the same random bits $s'_1, \ldots, s'_{\lfloor n/2 \rfloor}$ using both $S^{\mathsf{n\text{-}n}}$ and* RMStatSS*. We capture this task via* shareRandomValues *by taking the underlying linear secret sharing scheme $S := (S^{\mathsf{n\text{-}n}}, \mathsf{RMStatSS})$ to be the scheme that simultaneously shares a secret via both $S^{\mathsf{n\text{-}n}}$ and* RMStatSS *using independent randomness. Consequently, the ith share under S is a two-bit vector whose first (resp., second) entry is the ith share under $S^{\mathsf{n\text{-}n}}$ (resp., ith share under* RMStatSS*). The linear*

computation $([s'_1], \ldots, [s'_{\lfloor n/2 \rfloor}])^T = M \cdot ([s_{\pi(1)}], \ldots, [s_{\pi(n)}])^T$ *proceeds in the same way, where we think of* $([s_{\pi(1)}], \ldots, [s_{\pi(n)}])^T$ *as an* $n \times 2$ *matrix. Equivalently, the servers hold both* $[s_1]^{\mathsf{n\text{-}n}}, \ldots, [s_n]^{\mathsf{n\text{-}n}}$ *and* $[s_1]^{\mathsf{RM}}, \ldots, [s_n]^{\mathsf{RM}}$*, and locally compute* $([s'_1]^{\mathsf{n\text{-}n}}, \ldots, [s'_{\lfloor n/2 \rfloor}]^{\mathsf{n\text{-}n}})^T = M \cdot ([s_{\pi(1)}]^{\mathsf{n\text{-}n}}, \ldots, [s_{\pi(n)}]^{\mathsf{n\text{-}n}})^T$ *and* $([s'_1]^{\mathsf{RM}}, \ldots, [s'_{\lfloor n/2 \rfloor}]^{\mathsf{RM}})^T = M \cdot ([s_{\pi(1)}]^{\mathsf{RM}}, \ldots, [s_{\pi(n)}]^{\mathsf{RM}})^T$.

Efficiency. We note that the only communication in the protocol consists of sharing the secrets $s_1, \ldots, s_n$ via S. As for the computational complexity, each server P_i computes the shares $[s_i]$, and also has to locally compute the linear function M on its shares from $([s_{\pi(1)}], \ldots, [s_{\pi(n)}])$. We note that using the construction of [DI14] for a linear uniform output family that has linear-size circuits for left multiplication (see Theorem 3.10), we can also obtain linear-size circuits for right multiplication, see [DI14, Lemma 6]. Therefore, right multiplication by M can be executed in linear time. It is not hard to see that given a preprocessing stage of the permutation π, that generates a length-n list L with $L[i] = \pi(i)$, we can multiply M by the permuted values in time $O(n)$ in the RAM model. This preprocessing stage is executed only once, requires only $O(n \cdot \log(n))$ time in the RAM model, and can be used later in all the executions of shareRandomValues.[10]

5.1.2 The Basic Protocol

We continue with a description of our basic protocol, with offline/online phases for the case where an upper bound B on the number of instructions is known.[11] We then explain how this protocol can be extended for the general cases, where no upper bound on the number of instructions is known, and discuss some additional variants of the protocol.

Our protocol is parameterized by an $\lfloor n/2 \rfloor \times n$ matrix $M \in \mathbb{F}_2^{\lfloor n/2 \rfloor \times n}$, that is sampled once and for all, and is hardwired into the protocol. This matrix will be used for the sub-protocol shareRandomValues, and as discussed in Section 5.1.1, if we sample it from the $(2, \lfloor n/2 \rfloor, n)$-linear uniform output family of [DI14], then M is "good" with probability at least $1 - 2^{-\epsilon \cdot n/2}$. Therefore, we always assume that M is indeed good. We continue with an overview of our protocol.

Initialization. Already in the offline phase, we let the parties generate a random permutation π for the Reed-Muller secret sharing scheme and for the sub-protocol shareRandomValues. This step is executed by letting P_1 pick a random permutation π and broadcast it. Since the set of (passively) corrupt parties Corrupt was chosen by the adversary at the beginning of the execution, it holds that with probability $1 - 2^{-\Omega(\sqrt{n})}$ over the choice of π both correctness and privacy for the Reed-Muller secret sharing scheme RMStatSS hold, and with probability $1 - 2^{-\epsilon \cdot n/2}$ over the choice of π the sub-protocol shareRandomValues with parameters M and π can be used to generate shares of random values. Let us condition on this event.

The invariant. We denote the values of the registers $r_1, \ldots, r_R$ by $v_1, \ldots, v_R$, where initially $R = m$ and $v_1 = x_1, \ldots, v_m = x_m$. Throughout, we keep the invariant that v_i is shared among the

[10] In fact, if for every $m \leq n$ the RAM model allows us to pick a random integer between 1 to m in time $O(1)$, then using (modern versions of) the Fisher-Yates shuffle [FY38], the preprocessing time can be reduced to $O(n)$. See [Knu14] for more information.

[11] Formally, we assume that the ideal functionality $\mathcal{F}_{\mathsf{RT\text{-}MPC}}$ also contains a public parameter B for the upper bound on the number of instructions. The functionality counts the instructions it receives from the client and aborts after B instructions.

servers using $[v_i]_\pi^{\mathsf{RM}}$.

To make sure that the invariant holds at the beginning of the online phase, we let the servers execute an off-the-shelf MPC protocol to securely translate the n-out-of-n secret sharing $[x_1]^{\mathsf{n\text{-}n}}, \ldots, [x_m]^{\mathsf{n\text{-}n}}$ that they get as an input, into shares $[x_1]_\pi^{\mathsf{RM}}, \ldots, [x_m]_\pi^{\mathsf{RM}}$ of the Reed-Muller secret sharing. This step requires communication and computation of $\mathrm{poly}(n, m)$ which is independent of the number of instructions B, which we take to be much larger than m.[12]

As in most MPC protocols, performing linear operations over the registers is simple, and requires no communication, since the underlying secret sharing scheme is linear. Therefore, we continue by discussing the multiplication of two registers.

Multiplying two registers. Assume that the servers are instructed to multiply v_i and v_j and store the result in r_k. The invariant guarantees that the servers hold $[v_i]_\pi^{\mathsf{RM}}$ and $[v_j]_\pi^{\mathsf{RM}}$ and our goal is to compute a fresh sharing of $[v_i \cdot v_j]_\pi^{\mathsf{RM}}$. We denote the ℓth share of v_i (resp., v_j) by $v_{i,\ell}$ (resp., $v_{j,\ell}$). To compute the multiplication, we follow (a variant of) the technique of [BH08].

Let us first assume that a random bit s is distributed among the servers both using the Reed-Muller secret sharing scheme $[s]_\pi^{\mathsf{RM}}$, and using the n-out-of-n secret sharing scheme $[s]^{\mathsf{n\text{-}n}}$. We denote the shares by $(s_1^{\mathsf{RM}}, \ldots, s_n^{\mathsf{RM}})$ and $(s_1^{\mathsf{n\text{-}n}}, \ldots, s_n^{\mathsf{n\text{-}n}})$, respectively. Later, we will explain how the servers can generate the shares efficiently in the offline phase.

To compute the multiplication, we recall that if we multiply the shares of v_i and v_j, then the vector $(v_{i,1} \cdot v_{j,1}, \ldots, v_{i,n} \cdot v_{j,n})$ correspond to a (non-randomized) n-out-of-n secret sharing of $v_i \cdot v_j$. (See the discussion about the multiplication properties of the scheme in Section 3.2.) We therefore let every P_ℓ send $v_{i,\ell} \cdot v_{j,\ell} + s_\ell^{\mathsf{n\text{-}n}}$ to P_1. We observe that the vector $(v_{i,1} \cdot v_{j,1} + s_1^{\mathsf{n\text{-}n}}, \ldots, v_{i,n} \cdot v_{j,n} + s_n^{\mathsf{n\text{-}n}})$ is a randomized n-out-of-n secret sharing of $v_i \cdot v_j + s$, so it reveals no information other than $v_i \cdot v_j + s$, which is just a random bit, since s is uniformly distributed. We then simply let P_1 recover the bit $b := v_i \cdot v_j + s$ and broadcast b. Given this bit, the servers locally compute $[v_i \cdot v_j]_\pi^{\mathsf{RM}} := [s]_\pi^{\mathsf{RM}} + [b]_\pi^{\mathsf{RM}}$, where $[b]_\pi^{\mathsf{RM}}$ corresponds to some canonical sharing of $b_{i,j}$ under $\mathsf{RMStatSS}$ with public randomness π. (E.g., the all $b_{i,j}$ vector that corresponds to the constant multivariate polynomial $b_{i,j}$).

We observe that this step requires each server to communicate exactly 1 bit![13] In addition, the total computation time of this step is $O(n)$ in the RAM model.

Generating shares of random variables. We continue by explaining how to generate shares of random bits already in the offline phase. Since we have an upper bound B on the number of multiplication gates, our goal is to share B random bits $s'_1, \ldots, s'_B$ using both the Reed-Muller secret sharing and the n-out-of-n secret sharing. It is enough to explain how to share $\lfloor n/2 \rfloor$ random values among the servers, since sharing B values can be done by repeating this process (in parallel) for $B/\lfloor n/2 \rfloor$ times. To share $\lfloor n/2 \rfloor$ random bits, we simply execute $\mathsf{shareRandomValues}$ with the secret sharing scheme $S = (S^{\mathsf{n\text{-}n}}, \mathsf{RMStatSS})$, see Remark 5.5 for more details.

Observe that sharing B random values among the servers requires communication complexity of at most $4(1 + \frac{2}{n})Bn + 2n^2$ bits. One can also verify that the total computational complexity of

[12] We mention that this procedure can be executed efficiently using the techniques developed in this work. See Remark 5.7 for more details.

[13] If a broadcast channel is not available, then every server communicates 1 bit, except for P_1 that communicates $n - 1$ bits. We can balance this cost across many multiplications by letting each party take the role of P_1 once in every n multiplications. The amortized communication cost of each party per multiplication gate become $1 + 1/n$.

the servers is $O(B \cdot n \cdot \log(n))$ in the RAM model, due to the cost of generating Reed-Muller shares.

Randomizing outputs. To simplify the simulator, when the jth operation of the client is to reveal the value of r_i, we let the servers send the shares of $[v_i]_\pi^{\mathsf{RM}} + [s'_j]_\pi^{\mathsf{RM}} + [s'_j]^{\mathsf{n\text{-}n}}$, that form a randomized n-out-of-n secret sharing of v_i.

The protocol. The protocol appears in Figure 3.

Protocol RT-MPC

Public parameters: Positive integer m denoting the number of inputs. An $\lfloor n/2 \rfloor \times n$ matrix $M \in \mathbb{F}^{\lfloor n/2 \rfloor \times n}$. A positive integer B, denoting an upper bounds on the number of instructions.

Offline phase:

1. *(Generate public randomness)* The first server P_1 broadcasts a random permutation π of $\{1, \ldots, n\}$. The permutation is used both as a public randomness for the Reed-Muller secret sharing RMStatSS, as well as a public parameter in the execution of the sub-protocol shareRandomValues .
2. *(Generate shared randomness)* Let $L = \lceil B / \lfloor n/2 \rfloor \rceil$. For every $j = 1, \ldots, L$, the servers execute shareRandomValues with the secret sharing scheme $S = (S^{\mathsf{n\text{-}n}}, \mathsf{RMStatSS})$ (see Remark 5.5) to generate shares $[s'_{1,j}]_\pi^{\mathsf{RM}}, \ldots, [s'_{\lfloor n/2 \rfloor, j}]_\pi^{\mathsf{RM}}$, and $[s'_{1,j}]^{\mathsf{n\text{-}n}}, \ldots, [s'_{\lfloor n/2 \rfloor, j}]^{\mathsf{n\text{-}n}}$. We emphasize that in all execution we use the matrix M and permutation π as the public parameters of shareRandomValues.

   Observe that this step generates $L \cdot \lfloor n/2 \rfloor \geq B$ shares of random values. To simplify notation, we denote by $[s'_1]_\pi^{\mathsf{RM}}, \ldots, [s'_B]_\pi^{\mathsf{RM}}$, and $[s'_1]^{\mathsf{n\text{-}n}}, \ldots, [s'_B]^{\mathsf{n\text{-}n}}$ the first B shared values that are generated in this step.

Online phase:

1. *(Inputs)* The ith server holds inputs $x_{1,i}, \ldots, x_{m,i} \in \mathbb{F}_2$, that correspond to the ith shares of $[x_1]^{\mathsf{n\text{-}n}}, \ldots, [x_m]^{\mathsf{n\text{-}n}}$. The servers execute an MPC protocol Π (say, that of [BGW88]) to translate the shares $[x_1]^{\mathsf{n\text{-}n}}, \ldots, [x_m]^{\mathsf{n\text{-}n}}$ into fresh shares of $[x_1]_\pi^{\mathsf{RM}}, \ldots, [x_m]_\pi^{\mathsf{RM}}$.
2. *(Computing instructions)* Consider $R = m$ virtual registers, denoted $r_1, \ldots, r_R$, where at first x_i is stored at r_i. Throughout we keep the invariant that the value of r_i, denoted v_i is shared among the servers using $[v_i]_\pi^{\mathsf{RM}}$. This is true at the beginning, since for every $i = 1, \ldots, m$ the servers hold $[x_1]_\pi^{\mathsf{RM}}, \ldots, [x_m]_\pi^{\mathsf{RM}}$.

   The servers locally initialize a counter for number of the current instruction number, which is updated after the execution of every instruction. The servers also locally initialize a counter for the number of registers R.

   Upon receiving an instruction inst from the client, the servers do as follows.

   (a) If $\mathsf{inst} = (\mathsf{add}, i, j, k)$ for $i, j, k \in \{1, \ldots, R\}$, then the servers locally compute $[v_k]_\pi^{\mathsf{RM}} := [v_i]_\pi^{\mathsf{RM}} + [v_j]_\pi^{\mathsf{RM}}$.

   (b) If $\mathsf{inst} = (\mathsf{add1}, k)$ for $k \in \{1, \ldots, R\}$, then servers locally compute $[v_k]_\pi^{\mathsf{RM}} := [v_k]_\pi^{\mathsf{RM}} + [1]_\pi^{\mathsf{RM}}$, where $[1]_\pi^{\mathsf{RM}}$ corresponds to the shares defined by the constant polynomial 1.

   (c) If $\mathsf{inst} = (\mathsf{multiply}, i, j, k)$ for $i, j, k \in \{1, \ldots, R\}$, then every P_ℓ does as follows.

      i. Let c be the current instruction number (as per the instruction counter).

      ii. Let $v_{i,\ell}$ and $v_{j,\ell}$ be the shares of P_ℓ in $[v_i]_\pi^{\mathsf{RM}}$ and $[v_j]_\pi^{\mathsf{RM}}$, respectively. Let $s'_{c,\ell}$ be the share of P_ℓ in $[s'_c]^{\mathsf{n\text{-}n}}$.

iii. P_ℓ sends $b_\ell := v_{i,\ell} \cdot v_{j,\ell} + s'_{c,\ell}$ to P_1.

Upon receiving $b_1, \ldots, b_n$ from the servers, P_1 computes $b := b_1 + \ldots + b_n$ and broadcasts b. The servers then set $[v_k]_\pi^{\mathsf{RM}} := [s'_c]_\pi^{\mathsf{RM}} + [b]_\pi^{\mathsf{RM}}$, where $[b]_\pi^{\mathsf{RM}}$ corresponds to the shares defined by the constant polynomial b.

(d) If inst $=$ increase then the servers increase the counter of R by 1, and add an additional virtual server. The value of the new register is 0, and the servers hold a default sharing of $[0]_\pi^{\mathsf{RM}}$ defined by the constant polynomial 0.

(e) If inst $=$ (output, i) for $i \in \{1, \ldots, R\}$, and c is the current instruction number, then the servers send $[v_i]_\pi^{\mathsf{RM}} + [s'_c]_\pi^{\mathsf{RM}} + [s'_c]^{\mathsf{n\text{-}n}}$ to the client. The client uses $\mathsf{Recover}^{\mathsf{n\text{-}n}}$ to recover the value v_i from the shares, and outputs v_i.

Figure 3: Protocol RT-MPC

Extension to any number of parties. The construction can be extended to any number of parties $2^\alpha \leq n \leq 2^{\alpha+1} - 2$ by picking a random committee of size $2^\alpha - 1$ to execute the protocol, since the random committee contains at least $(\frac{1}{2} + \frac{\epsilon}{2})$-fraction of honest servers with probability $1 - 2^{-\Omega_\epsilon(n)}$. See further discussion in Section 5.1.3. We therefore obtain the following theorem.

Theorem 5.6. *Let $\epsilon > 0$, let n be the number of servers and let $t = (\frac{1}{2} - \epsilon)n$. Then protocol* RT-MPC *is a UC-secure realization of $\mathcal{F}_{\mathsf{RT\text{-}MPC}}$ with a bound B on the number of instructions, with statistical security and error $2^{-\Omega_\epsilon(\sqrt{n})}$ against a static, passive adversary that can corrupt the client and up to t servers.*

The total communication complexity in the offline phase is $O(Bn + mn + n^2)$ bits, and in the online phase is $Bn+mn$ bits. The total computational complexity in the offline phase is $O(Bn \log(n)+mn \log(n))$ and in the online phase is $O(Bn + mn)$.

By "total communication complexity" (resp., "total computational complexity") we mean the total number of bits sent in the protocol (resp., the total number of RAM-operations performed in the protocol), taken over all the parties.

Proof sketch. We continue with a proof of security for our protocol (the efficiency of the protocol is analysed below). Since our protocol follows the approach of [BGW88], with multiplication computed using the technique of [BH08], it provides perfect security conditioned on the events that (1) the secret sharing scheme RMStatSS with permutation π has perfect correctness and privacy, and (2) the sub-protocol shareRandomValues with public parameters (M, π) securely generates shares of random values. To bound the error probability, we note that the matrix M is good (event E_0) with probability $1 - 2^{-\epsilon \cdot n/2}$. Conditioned on this event, we notice that for every choice of the set Corrupt of corrupt parties it holds that (1) the secret sharing scheme RMStatSS provides perfect security (event E_1) with probability $1 - 2^{-\Omega(\sqrt{n})}$ over the choice of π, and (2) the sub-protocol shareRandomValues with public parameters (M, π) (event E_2) securely generates shares of random bits with probability $1 - 2^{-\epsilon \cdot n/2}$ over the choice of π. Therefore, the total error of our protocol is $2^{-\Omega(\sqrt{n})}$.

For completeness, we describe a simulator that achieves perfect security conditioned on the event $E_0 \wedge E_1 \wedge E_2$. For simplicity, we focus on the case where $n = 2^\alpha - 1$. (The generalization to any number of parties n is straightforward.)

Offline phase. In the offline phase, the simulator picks a random permutation π of $\{1,\ldots,n\}$, and sets π to be the broadcast message of P_1. For $j = 1,\ldots,L$, to simulate the $j$the execution of shareRandomValues we simply let the simulator execute the protocol honestly. That is, for every $i \in \mathsf{Honest}$ the simulator picks a random bit $s_{i,j}$, computes a fresh secret sharing $[s_{i,j}]^{\mathsf{RM}}_{\pi}$ and $[s_{i,j}]^{\mathsf{n\text{-}n}}$, and gives the corrupt servers their corresponding shares. The simulator executes the corrupt servers honestly to obtain the shares that they generate in the jth execution, as well as their shares of the extracted randomness $s'_{1,j},\ldots,s'_{\lfloor n/2\rfloor,j}$. Following the notation of the protocol, let us denote by $s'_1,\ldots,s'_B$ the first B random bits that are generated in this process.

Online phase. At the beginning of the online phase, the environment picks the inputs of the parties. The simulator receives the inputs of the corrupt servers $(x_{i,j})_{i\in\{1,\ldots,m\},j\in\mathsf{Corrupt}}$. The initialization process, in which the servers execute an MPC protocol Π to translate the shares $[x_1]^{\mathsf{n\text{-}n}},\ldots,[x_m]^{\mathsf{n\text{-}n}}$ into fresh shares $[x_1]^{\mathsf{RM}}_{\pi},\ldots,[x_m]^{\mathsf{RM}}_{\pi}$ is simulated as follows. First, we set $x'_i = 0$ for all $i = 1,\ldots,m$, and sample fresh sharing $[x'_i]^{\mathsf{RM}}_{\pi}$. Then we execute the simulator of Π on the inputs of the corrupt servers $(x_{i,j})_{i\in\{1,\ldots,m\},j\in\mathsf{Corrupt}}$, and the outputs of the corrupt servers in Π, that we set to be their shares in $[x'_1]^{\mathsf{RM}}_{\pi},\ldots,[x'_m]^{\mathsf{RM}}_{\pi}$.

At this point the simulator holds the Reed-Muller shares of the corrupt servers for the values $v_1,\ldots,v_R$ of the registers, that we denote by $(v_{i,j})_{i\in R,j\in\mathsf{Corrupt}}$. The simulator also holds the shares of the corrupt servers for the values $s'_1,\ldots,s'_B$. We denote the shares of $[s'_i]^{\mathsf{n\text{-}n}}$ that the corrupt servers hold by $(\sigma^{\mathsf{n\text{-}n}}_{i,j})_{j\in\mathsf{Corrupt}}$, and the shares of $[s'_i]^{\mathsf{RM}}_{\pi}$ by $(\sigma^{\mathsf{RM}}_{i,j})_{j\in\mathsf{Corrupt}}$. We continue by explaining how to simulate the instructions of the client (that are chosen by the environment). After each step, the simulator holds the shares of the corrupt servers for the updated values $v_1,\ldots,v_R$. Throughout, we let the simulator hold a counter for the current number of instruction and the current number of registers.

For the instruction (add,i,j,k) we simply set $v_{k,\ell} = v_{i,\ell} + v_{j,\ell}$ for every $\ell \in \mathsf{Corrupt}$. Similarly, for the instruction $(\mathsf{add1},k)$ we simply update $v_{k,\ell}$ to be $v_{k,\ell}+1$ for every $\ell \in \mathsf{Corrupt}$. To simulate the instruction increase, we simply let the simulator update the counter for the number of registers $R \leftarrow R+1$, and also set all the shares of the new register to be 0, i.e., $v_{R,\ell} = 0$ for every $\ell \in \mathsf{Corrupt}$.

Assume that the cth instruction is $(\mathsf{multiply},i,j,k)$. The simulator computes the messages that the corrupt servers send to P_1 (i.e., the value $v_{i,\ell}\cdot v_{j,\ell} + \sigma^{\mathsf{n\text{-}n}}_{c,\ell}$ for every $\ell \in \mathsf{Corrupt}$) and send them on behalf of the corrupt servers. To simulate the broadcast of P_1, we split into cases.

- If P_1 is honest, then the simulator simulates its broadcast by picking a random bit b and setting b to be the broadcast. Then the simulator sets $v_{k,\ell} = b + \sigma^{\mathsf{RM}}_{c,\ell}$.
- If P_1 is corrupted, then the message from an honest P_ℓ to P_1 is simulated by a random bit. Given all the messages to P_1, the simulator computes the bit b and broadcasts it on behalf of P_1. The simulator sets $v_{k,\ell} = b + \sigma^{\mathsf{RM}}_{c,\ell}$ for every corrupt P_ℓ.

Finally, if the instruction is (output,i) and the client is corrupted, the simulator first receives the output $y \in \{0,1\}$. The simulator samples an n-out-of-n secret sharing of y, denoted $(y_1,\ldots,y_n)$ conditioned on $y_\ell = v_{i,\ell} + \sigma^{\mathsf{n\text{-}n}}_{i,\ell} + \sigma^{\mathsf{RM}}_{i,\ell}$ for every $\ell \in \mathsf{Corrupt}$. The simulator sends y_i to the client on behalf of P_i. This completes the description of the simulator. □

Remark 5.7 (Efficient initialization)**.** *At the beginning of the online phase, the servers translate the shares $[x_1]^{\mathsf{n\text{-}n}},\ldots,[x_m]^{\mathsf{n\text{-}n}}$ to shares $[x_1]^{\mathsf{RM}}_{\pi},\ldots,[x_m]^{\mathsf{RM}}_{\pi}$ using an arbitrary protocol Π. We note that using the techniques developed in this work, this task can be executed efficiently in the following way. Already in the*

offline phase, we let the servers generate m shares of random bits $\rho_1, \ldots, \rho_m$ both via $S^{\mathsf{n\text{-}n}}$ and RMStatSS*, by executing* shareRandomValues *with $S = (S^{\mathsf{n\text{-}n}}, \mathsf{RMStatSS})$ for $\lceil m/\lfloor n/2 \rfloor \rceil$ times (this is the same technique used to generate the shares of $s'_1, \ldots, s'_B$). In the online phase, for each x_i the servers send their shares of $[x_i]^{\mathsf{n\text{-}n}} + [\rho_i]^{\mathsf{n\text{-}n}}$ to P_1, and P_1 recovers $b_i = x_i + \rho_i$ and broadcasts b_i.*[14] *Finally, the servers locally compute $[x_i]^{\mathsf{RM}}_\pi := [\rho_i]^{\mathsf{RM}}_\pi + [b_i]^{\mathsf{RM}}_\pi$, where $[b_i]^{\mathsf{RM}}_\pi$ is some canonical sharing of b_i under* RMStatSS *with public randomness π. (E.g., the all b_i vector that corresponds to the constant multivariate polynomial b_i).*

This procedure requires communicating a total of $4(1+\frac{2}{n})mn + 2n^2$ bits in the offline phase, and mn bits in the online phase. The total computational complexity in the offline phase is $O(m \cdot n \cdot \log(n))$, due to the generation of the Reed-Muller shares, and in the online phase is $O(mn)$.

Efficiency. We continue with an analysis of the efficiency of our protocol, assuming that we follow the initialization procedure of Remark 5.7. In the offline phase, the total communication is at most $4(1+\frac{2}{n})Bn + 4(1+\frac{2}{n})mn + 4n^2 + O(n \cdot \log(n))$ bits, where B is the upper bound on the number of instructions. As we usually think of $B \gg n, m$, this means that the total offline communication overhead per instruction is $4(1+\frac{2}{n})n + o(1) = 4n + 8 + o(1)$. In the online phase, the initialization step requires communication of mn bits. In addition, the instructions add, add1 and increase require no communication. To execute the instruction multiply and output, every server communicates exactly 1 bit. Therefore the total online communication overhead per instruction is at most $n + o(1)$.

As for the computational complexity, in the offline phase each party performs $O(B \cdot \log(n) + m \cdot \log(n))$ operations in the RAM model, so the total computational complexity in the offline phase is $O(B \cdot n \cdot \log(n) + mn \cdot \log(n))$. In the online phase, each party performs a constant number of operations per input in the initialization, except for P_1 which performs $O(n)$ operations per input (summing up the bits and recovering the secret). Therefore, the initialization can be executed in total computational complexity of $O(mn)$. Similarly, every instruction in the online model can be executed in total computational complexity of $O(n)$, so the total computational complexity of the online phase is $O(Bn + mn)$. We conclude that the total offline computational overhead per instruction is $O(n \cdot \log(n))$ and the total online computation overhead per instruction is $O(n)$.

5.1.3 Variants of the Basic Protocol

Protocol with unbounded number of instructions. So far we assumed that there is an upper bound B on the number of instructions. We observe that the only place where we use the upper bound B is the generation of the shared random bits $[s'_1]^{\mathsf{RM}}_\pi, \ldots, [s'_B]^{\mathsf{RM}}_\pi$ and $[s'_1]^{\mathsf{n\text{-}n}}, \ldots, [s'_B]^{\mathsf{n\text{-}n}}$. When there is no upper bound on the number of instructions, we simply generate these shares in the online phase, on-the-fly. That is, we let the servers generate $\lfloor n/2 \rfloor$ shares of random bits, by letting each of them share a single random bit, and then execute shareRandomValues. Those random bits will be used for the computation of $\lfloor n/2 \rfloor$ multiplication instructions, after which we repeat the process. We note that the total communication and computation complexity of the protocol remains the same. However, since some of the work is postponed to the online phase, the exact communication and computation complexity of the online phase grow. Indeed, the online total communication overhead per instruction is $5n + 8 + o(1)$ bits, and the total computational overhead per instruction is $O(n \cdot \log(n))$.

[14] Like in Footnote 13, this task can be balanced across all the parties.

Improving efficiency by picking a committee. Following the technique of Bracha [Bra87], we observe that we can improve the efficiency of the protocol picking a random committee A of n' servers, and let only the servers inside the committee to execute the protocol. With probability at least $1-2^{-\epsilon^2 \cdot n'/2}$, the committee A contains at least $(\frac{1}{2}+\frac{\epsilon}{2})$-fraction of honest servers, and therefore security holds.

More formally, at the beginning of the execution we let P_1 broadcast a random committee A of n' servers. We identify the servers in the committee as n' servers $(P'_1, \ldots, P'_{n'})$ and execute the n'-party protocol with $(P'_1, \ldots, P'_{n'})$. This approach is applicable for all parts of the protocol, except the initialization step, in which the servers translate the shares $[x_1]^{\text{n-n}}, \ldots, [x_n]^{\text{n-n}}$ to Reed-Muller shares. Indeed, in this step the shares of all the servers are required in order to recover $x_1, \ldots, x_n$ and re-share them using the Reed-Muller scheme. Therefore, this is the only phase where all servers participate in the execution, but the new Reed-Muller shares that are generated in this step are n'-party Reed-Muller secret sharing that are given only to the parties in the committee A. We emphasize that this step is independent of the number of instructions B, and therefore does not affect the overhead of the protocol.

The error probability of the n'-party protocol is $2^{-\Omega(\sqrt{n'})}$ and therefore we can take $n' = \log^{2+\delta}(n)$ for any $\delta > 0$ and still maintain negligible error. We note that the total communication overhead per instruction in the offline phase is reduced to $(4\log^{2+\delta}(n) + 8) + o(1)$ bits, and the total communication overhead per instruction in the online phase is $\log^{2+\delta}(n) + o(1)$ bits. As for the computational complexity, the total overhead per instruction in the offline phase is $O(\log^{2+\delta}(n) \cdot \log\log(n))$ and in the online phase it is $O(\log^{2+\delta}(n))$.

References

[ANP23] Benny Applebaum, Oded Nir, and Benny Pinkas. How to recover a secret with o(n) additions. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 236–262. Springer, 2023.

[BGK20] Andrej Bogdanov, Siyao Guo, and Ilan Komargodski. Threshold secret sharing requires a linear-size alphabet. *Theory Comput.*, 16:1–18, 2020.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2008.

[BI01] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 912–926. Springer, 2001.

[Bra87] Gabriel Bracha. An o(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19. ACM, 1988.

[CCG+07] Hao Chen, Ronald Cramer, Shafi Goldwasser, Robbert de Haan, and Vinod Vaikuntanathan. Secure computation from random error correcting codes. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference*

on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings, volume 4515 of *Lecture Notes in Computer Science*, pages 291–310. Springer, 2007.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 395–426. Springer, 2018.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[CDN15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[CGH+85] Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 396–407. IEEE Computer Society, 1985.

[DI14] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 169–182. ACM, 2014.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.

[DNPR16] Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael A. Raskin. On the communication required for unconditionally secure multiplication. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 459–488. Springer, 2016.

[FY38] RA Fisher and F Yates. Statistical tables for biological, agricultural and medical research. 1938.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and ohn

A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710. ACM, 1992.

[GIOZ17] Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 420–446. Springer, 2017.

[GIP15] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 721–741. Springer, 2015.

[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 V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 275–304. Springer, 2021.

[GRS23] Venkatesan Guruswami, Atri Rudra, and Madhu Sudan. Essential coding theory. 2023. draft available at https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/.

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 172–190. Springer, 2005.

[HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. Ot-combiners via secure computation. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 393–411. Springer, 2008.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2005.

[HP03] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

[IMSW14] Yuval Ishai, Hemanta K. Maji, Amit Sahai, and Jürg Wullschleger. Single-use ot combiners with near-optimal resilience. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*, pages 1544–1548. IEEE, 2014.

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

[KB80] Rob Kaas and Jan M Buhrman. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, 34(1):13–18, 1980.

[KKM+17] Shrinivas Kudekar, Santhosh Kumar, Marco Mondelli, Henry D. Pfister, Eren Sasoglu, and Rüdiger L. Urbanke. Reed-muller codes achieve capacity on erasure channels. *IEEE Trans. Inf. Theory*, 63(7):4298–4316, 2017.

[Knu14] Donald E Knuth. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison-Wesley Professional, 2014.

[LXYY23] Hongqing Liu, Chaoping Xing, Yanjiang Yang, and Chen Yuan. Ramp hyper-invertible matrices and their applications to MPC protocols. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 204–236. Springer, 2023.

[Mas93] James L Massey. Minimal codewords and secret sharing. In *Proceedings of the 6th joint Swedish-Russian international workshop on information theory*, pages 276–279, 1993.

[Mas95] James L Massey. Some applications of coding theory in cryptography. *Codes and Ciphers: Cryptography and Coding IV*, pages 33–47, 1995.

[MS81] Robert J. McEliece and Dilip V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 73–85. ACM, 1989.

[Rot06] Ron M. Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

A Appendix: Comparison with Ramp Secret Sharing

In this section we compare our scheme to concrete constructions of ramp secret sharing from AG codes and random linear codes.

Comparison with AG codes. A $(t, t+\epsilon n)$-ramp linear secret sharing scheme with constant size alphabet $O(1/\epsilon^2)$ can be constructed from AG codes (see [CC06]). For every $t < (\frac{1}{2}-\epsilon)n$ the scheme is multiplicative, and for $t < (\frac{1}{3}-\frac{2\epsilon}{3})n$ the scheme is even *strongly multiplicative*[15]. Compared to AG codes, our constructions achieve a share size of 1 bit by using only simple algebraic techniques. This simplicity allows us to have efficient sharing procedure with linear or quasi-linear time. To the best of our knowledge, sharing a secret in an AG-codes scheme requires $O(n^2)$ operations.

Comparison with random linear codes. Using the fact that random linear codes achieve the Gilbert-Varshamov bound, it is possible to construct a $(t, t+\epsilon n)$-ramp linear secret sharing from linear codes over some constant-size alphabet that depends on t, ϵ, with probability $1 - 2^{-\Omega(n)}$. In fact, the work of Druk and Ishai [DI14] constructs such schemes with a sharing function that has a *linear-size circuit*. Insisting on binary shares, and, say, $0.49n$-privacy, this method can only obtain a $(0.49n, 0.999n)$-ramp secret.

Unfortunately, these schemes are not multiplicative. Multiplicative schemes can be achieved using the techniques of [CDM00, CCG+07], by combining shares from a linear code and its dual. In particular, [CCG+07] prove that for every $\epsilon > 0$, and every integer m, there exists a finite field $\mathbb{F}_q$ for $q = O_{\epsilon,m}(1)$, such that a random linear code over $\mathbb{F}_q$ is $((\frac{1}{2}-\epsilon)n, \frac{1}{2}n)$-ramp linear secret sharing with probability $1 - 2^{-m}$. However, using this technique they cannot achieve linear-size circuits for the sharing function. In addition, if we insist on binary shares, this method can only achieve a $(0.1n, 0.9n)$-ramp secret sharing.

Summary. The results are summarized in the following table. The first row corresponds to our first construction, that is based on Reed-Muller codes, and the second row correspond to the construction based on random linear codes and the technique of [DI14]. For simplicity, in the error probability we count both the initialization error of randomized constructions, as well as the error parameter of our t-static secret sharing.

[15] A $(t, t+\epsilon n)$-ramp linear secret sharing scheme is strongly multiplicative if for any set I of size $n-t$, and any two secret s and s' that are shared via the sharing vectors $(s_1, \ldots, s_n)$ and $(s'_1, \ldots, s'_n)$, respectively, the multiplication $s \cdot s'$ can be computed as a linear combination of the values $(s_i \cdot s'_i)_{i \in I}$.

Scheme	Type	Parameters	Share Size (Bits)	Sharing Time	Error Probability	Multiplicative?	Strongly Multiplicative?
Construction 1	t-static	$t = pn$ $0 < p < 1/2$	1	$O(n \cdot \log(n))$	$2^{-\Omega_p(\sqrt{n})}$	Yes	No
Construction 2*	t-static	$t = pn$ $0 < p < 1/2$	1	$O(n)^{\S}$	$2^{-\Omega_p(n)}$	No	No
Shamir [Sha79]	Threshold	$t = t_p = t_c - 1$	$O(\log(n))$	$O(n \cdot \log(n))$	-	for $t < n/2$	for $t < n/3$
AG Codes [CC06]	ramp	$t_p = t$ $t_c = t + \epsilon n$	$O(\log(1/\epsilon))$	$O(n^2)$	-	for $t < (\frac{1}{2} - \epsilon)n$	for $t < (\frac{1}{3} - \frac{2\epsilon}{3})n$
Random Linear Codes [DI14]*	ramp	$t_p = t$ $t_c = t + \epsilon n$	$O_{\frac{t}{n},\epsilon}(1)$	$O(n)$	$2^{-\Omega(n)}$	No	No
Random Linear Codes [CCG$^+$07]*	ramp	$t_p = (\frac{1}{2} - \epsilon)n$ $t_c = \frac{1}{2}n$	$O_{\epsilon,m}(1)$	$O(n^2)$	2^{-m}	Yes	No

* Randomized construction.
§ Requires preprocessing time of $O(n \cdot \log(n))$.

Table 1: Comparison of secret sharing schemes

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il