Threshold Garbled Circuits with Low Overhead

Schuyler Rosefield abhi shelat LaKyah Tyner
rosefield.s@northeastern.edu abhi@neu.edu tyner.l@northeastern.edu

February 23, 2024

Abstract

The folklore approach to designing a threshold variant of symmetric cryptographic algorithms involves applying generic MPC methods to secret sharing techniques: the MPC first combines participant input shares using the secret sharing scheme, and then evaluates the cryptographic function on the reconstructed key. Hardening this secure against $n-1$ malicious parties requires some mechanism to ensure input consistency, e.g., adding MACs to inputs, which consequently, increases the number of inputs and gates to the MPC. In many cases, this extra overhead is substantially more than the underlying cost of evaluating the symmetric cryptographic algorithm.

We present a scheme that can convert any suitable maliciously secure dishonest majority boolean-circuit $\mathcal{F}_{\mathsf{MPC}}$ into a threshold scheme $\mathcal{F}_{\mathsf{thresh}}$ with *almost no overhead*. Specifically, we present an SUC-secure scheme that allows for reactive threshold t-of-n boolean circuit evaluation amongst a group of n parties P, for any $t \leq n$, against a malicious adversary that corrupts any number of parties less than the threshold t. Moreover, multiple circuits can be evaluated sequentially with the secret-shared authenticated outputs of a circuit to be used subsequently as inputs for a new circuit by any $S \subseteq P$ of size $|S| \geq t$.

Building upon the works of Wang et al, Hazay et al, and Yang et al, [WRK17, HSSV17, YWZ20] for dishonest majority $\mathcal{F}_{\mathsf{MPC}}$, our key insight is to create *threshold versions* of the "authenticated bits" used to handle input in these recent n-party garbled circuits protocols. The resulting design incurs a small overhead to produce the reusable "threshold authenticated bits" during preprocessing, and adds no extra communication to evaluate with the authenticated input during the online phase.

Using our methods, thresholdizing a boolean circuit has essentially no performance overhead. For example, to compute HMAC, a full Setup+Eval execution of the $(n-2)$-out-of-n thresholdized version is approximately 4% more expensive than the state-of-the-art n-party MPC. In contrast, using the folklore method is approximately 100% more expensive. This is especially true for small circuits such as AES which has 6800 gates and thus incurs the most overhead for thresholdizing. Simply considering the online Eval cost, our approach can evaluate AES blocks at $2.3/s$ with 16 parties, exceeding the baseline MPC cost without preprocessing, and surpassing the folklore method that only achieves $.33/s$ blocks. Ultimately, this result makes threshold boolean circuit MPC as feasible as any MPC application.

1 Introduction

A threshold cryptography scheme enables a set of n parties to split a secret key so that a subset of k of the n parties can jointly perform operations using the key without explicitly reconstructing it. Recent specialized MPC techniques have made it feasible to design threshold schemes for legacy signature and encryption schemes such as ECDSA [LN18, GG18, DKLs19, CCL+20, KMOS21, CGG+20, ANO+22, oST23] and RSA [RSA78, DK01, Sho00, CDK+22]. These recent examples of efficient protocols exploit algebraic properties of the underlying schemes to design efficient threshold protocols. In contrast, there has been little progress in designing threshold schemes for symmetric cryptographic algorithms such as AES or SHA-X that have *limited* algebraic structure. We begin by discussing the *Folklore* method to create a threshold scheme using standard tools and difficulties in the boolean case.

Standard threshold secret sharing schemes are built using polynomial secret shares over a large field $\mathbb{F}_{2^s}$ that encode the secret as the 0 coordinate and assigns another unique polynomial evaluation to each party. To achieve malicious security, one can apply a SPDZ-style MACs in which the authenticated threshold secret share of a value $x \in \mathbb{F}_{2^s}$ consists of three components $\langle\langle x \rangle\rangle = ([\![\alpha]\!], [\![x]\!], [\![\alpha x]\!])$ where α is a shared global key.

The *Folklore* method for constructing a threshold MPC functionality takes a generic dishonest-majority boolean circuit $\mathcal{F}_{\mathsf{MPC}}$, and augments the desired circuit $\mathcal{C}(x)$ to take in and reconstruct *threshold* authenticated shares $\langle\langle x \rangle\rangle$ on the input, and if valid, output $\langle\langle \mathcal{C}(x) \rangle\rangle$. Doing so intrinsically adds overhead over calculating the base circuit $\mathcal{C}$ as additional gates and wires are required to reconstruct the shared value x, check the validity of the MAC αx, and to generate new authenticated shares as output $\langle\langle y \rangle\rangle$. In particular, as our experiments show, processing the secret shared inputs and performing authentication checks by compiling into the executed circuit *within* the MPC can be *much* more costly than evaluating the original circuit $\mathcal{C}$. In addition to overhead during evaluation, this Folklore method also induces overhead in a *setup* phase that is used to compute authenticated secret shares of the threshold secret key.

To quantify the overhead of this Folkmore method, we must carefully chose secret sharing parameters, and in particular the finite field, since the overhead essentially boils down to the circuit complexity of finite field multiplication. In particular, the field $\mathbb{F}_{2^s}$ has a boolean circuit for addition that requires 0 AND gates and a multiplication circuit that uses $O(s \log s)$ AND gates (using the Fast Fourier Transform to calculate polynomial multiplication). In contrast, although Harvey and van der Hoeven's integer multiplication also has the same asymptotic complexity, *modular* field multiplication in $\mathbb{F}_p$ for $p > 2$ requires both an asymptotically larger multiplication circuit, and a concretely much larger circuit (using more practical Karatsuba-style algorithms). Performing the MAC check in the boolean circuit requires one field multiplication per (field-size) input in the boolean circuit and an equality check, and requires a *multiplicative* increase in input wires that is proportional to the number of parties. Likewise generating a new MAC requires a field multiplication per field-sized output as well as the evaluation of a random polynomial on n points. Depending on the circuit description this $O(s \log s)$ gate increase per (field-size) input for MAC checking can be as costly as the actual application circuit such as AES, or an order of magnitude more expensive for simpler circuits such as Hamming distance. Altogether, for a fixed circuit $\mathcal{C} = (\mathcal{I}, G, \mathcal{O})$ with $\mathcal{I}$ input wires, G AND gates, and $\mathcal{O}$ output wires, this Folklore approach incurs an additive communication overhead of $O(n\kappa(n^2 + s \log s)\mathcal{O})$ in setup, and $O(n\kappa(n + s \log s)\mathcal{I})$ in eval.

1.1 Our Contribution

The main goal of this paper is to understand and improve upon the inherent overhead in implementing a threshold cryptosystem for arbitrary boolean circuits.

In contrast to the Folklore approach outlined above, we present a scheme that can convert any suitable maliciously secure dishonest majority boolean-circuit $\mathcal{F}_{\mathsf{MPC}}$ into a threshold scheme $\mathcal{F}_{\mathsf{thresh}}$ with *almost no overhead*.

Specifically, we present an SUC-secure scheme in the Random Oracle Model that allows for reactive threshold t-of-n boolean circuit evaluation amongst a group of n parties P, for any $t \leq n$, against a malicious adversary that corrupts any number of parties less than the threshold t. Multiple circuits can be evaluated sequentially with the secret-shared authenticated outputs of a circuit to be used subsequently as inputs for a new circuit by any $S \subseteq P$ of size $|S| \geq t$. This directly supports the pattern of running a re-usable *preprocessing phase* followed by any number of *evaluation phases*. In particular, our method *does not* follow the folklore pattern to evaluate a MAC or secret sharing reconstruction explicitly as a circuit within an MPC. The only communication overhead incurred in our approach is restricted to key setup where it is at most 10% among the circuits we selected for benchmarking, as low as .6%, and amortizes towards 0 across all evaluations.

We present our main theorem here informally.

Theorem 1.1 (Main Theorem (*Informal*))**.** *The protocol π_{thresh} SUC-realizes $\mathcal{F}_{\mathsf{thresh}}$ in the $(\mathcal{F}_{\mathsf{MPC}}, \cdot)$-hybrid model against a malicious adversary that statically corrupts up to $t-1$ parties with $O(n\kappa(\mathcal{O} + \kappa + s))$ additive communication overhead ($O(1)$ multiplicative) compared to $\mathcal{F}_{\mathsf{MPC}}$ in Setup, and 0 communication overhead in Eval.*

Roughly the additive overhead of our scheme is derived from Claim 5.1 to generate the threshold resharing, and the multiplicative overhead from Claim 5.2 being equivalent to the asymptotic complexity of the underlying $\mathcal{F}_{\mathsf{MPC}}$ scheme. Comparing results, it follows that the overhead of the folklore method is a factor of $(n^2 + s \log s)$ greater than our method in Setup, and of course significantly larger than constant in Eval.

To achieve our result, we extend the notion of *authenticated bits* used in the works of Wang et al, Hazay et al, and Yang et al, [WRK17, HSSV17, YWZ20] for dishonest majority n-party garbled circuits. An *authenticated bit* $\langle b \rangle$ (Def. 2.2) is an authenticated secret sharing scheme where each party holds an additive share of a bit b, and pairwise MACs on the shares of b. In the evaluation phase of the garbling schemes of [WRK17, HSSV17, YWZ20], the parties hold a random authenticated bit $\langle r_w \rangle$ for each wire w in the circuit $\mathcal{C}$, and calculate the masked value $\Lambda_w = x_w + r_w$ as well as the corresponding garbled labels. For an input wire w provided by $\mathcal{P}^i$ with value x_w, the public value Λ_w is calculated by running $\texttt{Open}(x_w + \langle r_w \rangle)$. Likewise for an output wire w given to $\mathcal{P}^i$, the parties open the mask $\langle r_w \rangle$ to $\mathcal{P}^i$ who can then calculate $x_w = \Lambda_w + r_w$. Immediately it is clear that the garbling scheme can be extended to use authenticated inputs and outputs, where an authenticated input is calculated $\Lambda_w = \texttt{Open}(\langle x_w \rangle + \langle r_w \rangle)$ and an authenticated output $\langle x_w \rangle = \Lambda_w + \langle r_w \rangle$.

Our main construction is a protocol π_{thresh} that can produce *threshold authenticated bits* $\langle\langle \cdot \rangle\rangle$ (Def. 2.3) such that any set of t parties can locally convert into t-party authenticated bits compatible with $\mathcal{F}_{\mathsf{MPC}}$. In this sense π_{thresh} contains a verifiable secret sharing sub-protocol for BDOZ-style [BDOZ11] authenticated shares. These threshold authenticated bits can either be sampled uniformly, or converted from non-threshold authenticated bits. This way multiple instances of the garbling protocol that output authenticated bits can be chained with a threshold-resharing procedure that ensures consistency between each garbled circuit.

To construct threshold authenticated bits, we first observe that the scheme for authenticated bits $\langle \cdot \rangle$ is linearly homomorphic and that many *standard* secret sharing schemes such as Shamir's have the property that reconstructing a secret shared value, which we denote by $[\![x]\!]$, is a linear operation. Abusing notation we consider authenticated bits of a t-of-n secret shared value $\langle [\![x]\!] \rangle$ to be the authenticated bits created where each party supplies the bit decomposition of its threshold share $\texttt{Bits}([\![x]\!]^i)$ as input. A set of parties S can then evaluate the reconstruction operation $\texttt{Recon}^S(\langle [\![x]\!] \rangle)$ as bit operations on the authenticated bits. This constructs non-threshold authenticated bits $\langle x \rangle$ of the underlying value. In other words, $\texttt{Recon}^S(\langle [\![x]\!] \rangle)$ performs the following transformation $\langle [\![x]\!] \rangle \mapsto \langle x \rangle$. Moving forward, we will use the notation $\langle\langle x \rangle\rangle$ instead of $\langle [\![x]\!] \rangle$ to denote a *threshold* authenticated bit x.

Constructing a random value $\langle\langle x \rangle\rangle$ requires the parties to first hold t-of-n threshold shares $[\![x]\!]$ of a bit x and then authenticate the bit-decomposition of each share denoted $\boldsymbol{B}^i = \texttt{Bits}([\![x]\!]^i)$ to create $\langle \boldsymbol{B} \rangle$. The parties then run a consistency check on $\langle \boldsymbol{B} \rangle$ to confirm that the authenticated bits represent evaluations of a valid degree $t - 1$ polynomial, and if so sets $\langle\langle x \rangle\rangle = \langle \boldsymbol{B} \rangle$. The case of re-sharing an existing $\langle x \rangle$ is done in an additional round by first sampling a random threshold share $\langle\langle r \rangle\rangle$, opening $\langle x \rangle + \langle r \rangle$, and calculating $\langle\langle x \rangle\rangle = (x + r) + \langle\langle r \rangle\rangle$.

In practice, the functionality $\mathcal{F}_{\mathsf{aBit}}$ that creates authenticated bits $\langle x \rangle$ is realized using a correlated OT extension protocol, and approximately requires ℓ pairwise extensions each of size κ to create ℓ threshold authenticated bits. Asymptotically in ℓ, this per-party cost is $O(n\ell\kappa)$, and the full cost analysis can be found in Section 5.

Next, we use this $\mathcal{F}_{\mathsf{aBit}}$ (itself a building block of $\mathcal{F}_{\mathsf{MPC}}$), in conjunction with a dishonest-majority maliciously secure $\mathcal{F}_{\mathsf{MPC}}$ (such as the garbling scheme of [YWZ20]), to construct threshold protocols for any boolean circuit. This is done without introducing any additional assumptions, and maintains the desired constant-round nature of the garbling protocols.

To illustrate our technique, we implement three applications, AES, HMAC-SHA2, and KMAC where the key material k (and associated preprocessing) is shared such that any t-of-n parties can evaluate $F_k(\cdot)$ without revealing k.

In contrast to the folklore construction of emulating a threshold cryptosystem within $\mathcal{F}_{\mathsf{MPC}}$, our construction for $\mathcal{F}_{\mathsf{thresh}}$ does not impart any increase on the circuit size. Intuitively this is because threshold authenticated bits can interact natively with $\mathcal{F}_{\mathsf{MPC}}$ after simple local operations bypassing the need to check authentication *within* the circuit being executed. Additionally, the communication overhead of $\mathcal{F}_{\mathsf{thresh}}$ over the generic non-threshold $\mathcal{F}_{\mathsf{MPC}}$ itself is minimal. The overhead during setup to create the threshold authenticated bits $\langle\langle \boldsymbol{x} \rangle\rangle$ is approximately the same cost as authenticating the same number of non-threshold authenticated bits $\langle \boldsymbol{x} \rangle$. In fact, for a setup circuit $\mathcal{C}$ with ℓ output bits, $\mathcal{F}_{\mathsf{MPC}}$ necessarily makes ℓ authenticated bits regardless. Then even for a circuit $\mathcal{C}$ with 0 AND gates the communication during setup in $\mathcal{F}_{\mathsf{thresh}}$ is at most 2x the cost of running $\mathcal{F}_{\mathsf{MPC}}$ without the threshold resharing.

Concretely, for the case of key setup for AES ($|\mathcal{C}| = 1360, \ell = 1408$) and KMAC ($|\mathcal{C}| = 38400, \ell = 1600$), this communication overhead is 10% and 0.6% respectively where $|\mathcal{C}|$ is the number of AND gates and ℓ is the size of the output in bits (see Fig. 1). Further, there is essentially *zero* overhead during the evaluation phase of $\mathcal{F}_{\mathsf{thresh}}$ over $\mathcal{F}_{\mathsf{MPC}}$ as parties can locally convert their threshold authenticated bits $\langle\langle x \rangle\rangle$ to t-party authenticated bits $\langle x \rangle$, and the cost of evaluating $\mathcal{C}$ is exactly the cost of evaluating $\mathcal{F}_{\mathsf{MPC}}$ with t parties. We confirm this through an implementation and concrete analysis in §5.

1.2 Related Work

Several prior works present MPC protocols for securely evaluating boolean circuits [BMR90, DPSZ12, LPSY19, BLN$^+$21, WRK17, AFSH$^+$20]. In general, most of these take inspiration from either Yao's Garbled Circuits [Yao86] or the Goldreich, Micali, and Wigderson protocol [GMW87]. Damgård et al. (SPDZ), for example, takes after the latter and is a general MPC protocol for computing arithmetic circuits [DPSZ12, DKL$^+$12, KOS16, ACE$^+$21], while other are able to support the use of boolean circuits [LOS14, FKOS15, AFSH$^+$20, CG20].

Many distributed garbling schemes operate in the online/offline model. In the offline (or preprocessing) phase the parties learn some information about the function or circuit that will be computed. In the online phase, each party learns their input and the output of the function is computed. Prior works such as [LPSY19, WRK17, HSSV17, HIV17, KRRW18, ZCSH18, YWZ20, BECO$^+$21, EXY22, HKO23] present efficient constant round protocols, many of which are malicious secure against an adversary corrupting any number of parties. Yang, Wang, and Zhang [YWZ20] presented a constant round protocol which now serves as the state-of-the art in multiparty setting. It improved upon the efficiency of prior work by developing new methods of generating multiparty authenticated bits and shares. In the maliciously-secure two party setting, the recent distributed garbling works of [KRRW18, DILO22, CWYY23] achieve efficiency close to semi-honest half gates in the authenticated garbling framework.

The most direct relevant work is the universal thresholdizer introduced by Boneh et. al. [BGGK17], which is a fully homomorphic encryption based framework for augmenting many cryptographic schemes to support threshold functionality. Their scheme is secure under the Learning with Errors (LWE) assumption. The framework makes it possible to construct a single round threshold scheme from any non-threshold signature or encryption scheme. A follow-up work by Cheon et. al.[CCK23] presents a scheme to reduce the communication required for achieving the compactness property of the Universal Thresholdizer. They save a factor of approximately $O(N^2)$ for the communication of the [BGGK17] scheme. While these schemes achieve optimal round complexity, they are otherwise not practical.

2 Preliminaries

Notation Scalars x are represented by lower-case letters; vectors $\boldsymbol{v}$ and matrices $\boldsymbol{M}$ are bolded. The i^{th} entry of a vector is denoted v_i, and the i^{th} row of a matrix by $\boldsymbol{m}_i$. A multidimensional array with ordered dimensions may be sliced, e.g. $\boldsymbol{M}_{*,*,k}$ indicates the degree-2 array with fixed index k. Typically indexing is 1-based.

Parties are denoted as $\mathcal{P}^1, \dots, \mathcal{P}^n \in P$, and $\mathcal{H}, \mathcal{B} \subset P$ refer to the sets of indices of honest, corrupt, and all parties respectively. A party $\mathcal{P}^i$ that is assigned a value is denoted with a superscript x^i. Most commonly this is used to denote $\mathcal{P}^i$'s component of a secret-shared value, such as $[x]^i$. The variable κ represents the computational security parameter, and s the statistical security parameter.

Primarily operations are performed in the security parameter-sized field $\mathbb{F}_{2^\kappa}$. Where appropriate, values $b \in \{0,1\}$ are embedded in elements of $\mathbb{F}_{2^\kappa}$ in the natural way, and likewise, the function $\texttt{Bits}(\cdot) : \mathbb{F}_{2^\kappa} \to \{0,1\}^\kappa$ interprets a field element as its bit decomposition. The function $\texttt{Combine}(\cdot) : \mathbb{F}_{2^\kappa}^\kappa \to \mathbb{F}_{2^\kappa}$ is analogous to the inverse of $\texttt{Bits}(\cdot)$ in that it takes κ elements and combines them into a single field element. If the basis $\{1, \alpha, \alpha^2, \dots, \alpha^{\kappa-1}\}$ is used for bit decomposition, then $\texttt{Combine}(\boldsymbol{x}) = \sum_i \alpha^{i-1} x_i$.

Notation for authenticated bits We denote simple additive shares of field elements $x \in \mathbb{F}_{2^\kappa}$ as $[x]$ where $x = \sum_i [x]^i$. To denote polynomial Shamir "t-of-n

threshold shares" of a field element $x \in \mathbb{F}_{2^\kappa}$, we use $[\![x]\!]$.

It is possible to locally convert a threshold sharing $[\![x]\!]$ to an additive share $[x]$ amongst a group of parties S using Lagrange interpolation. We denote the Lagrange basis polynomial with a set of points S and $i \in S$ as $\lambda_i^S(x) : \mathbb{F}_{2^\kappa} \to \mathbb{F}_{2^\kappa} = \prod_{j \in S \setminus \{i\}} \frac{x-j}{i-j}$. Then for $S \subseteq P$ with $|S| \geq t$ and $i \in S$, $\mathcal{P}^i$'s component of the additive share can be computed as $[x]^i := \lambda_i^S(0) \cdot [\![x]\!]^i$. This exactly coincides with reconstructing a Shamir-shared secret. Additionally, we use $\langle x \rangle$ to denote an additive authenticated bit and $\langle\langle x \rangle\rangle$ to denote a threshold authenticated bit.

2.1 Definitions

Definition 2.1. Information-Theoretic Message Authention Code (MAC)

A MAC scheme is a tuple of algorithms ($\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify}$)

- $\texttt{KeyGen}(\kappa)$: This is a randomized algorithm that takes as input the security parameter, and outputs a key $\Delta \leftarrow \mathbb{F}_{2^\kappa}$.
- $\texttt{Sign}_\Delta(x)$: This is a randomized algorithm that is parameterized by a secret key Δ and takes as input a value x to produce an instance key $K[x] \in \mathbb{F}_{2^\kappa}$ and an *authentication code* $M[x] \in \mathbb{F}_{2^\kappa}$ on x.
- $\texttt{Verify}_\Delta(x, M[x]; K[x])$: This is an algorithm that is parameterized by a key and takes a value x, its MAC $M[x]$, and the signing randomness $K[x]$, and output $\top$ if $M[x]$ is valid under $\Delta, K[x]$ and $\bot$ otherwise.

A secure MAC scheme must satisfy the following properties

- Correctness: For each $x \in \mathbb{F}_{2^\kappa}$

$$\Pr_\Delta[\texttt{Verify}_\Delta(x, \texttt{Sign}_\Delta(x)) = \top] = 1$$

- Soundness: For each $x' \neq x$ and $M[x'] \in \mathbb{F}_{2^\kappa}$

$$\Pr_\Delta[\texttt{Verify}_\Delta(x', M[x'], K[x]) = \top] \leq \text{negl}(\kappa)$$

We define the scheme below.

Authenticated bits Following prior work in the literature [NNOB12, WRK17, HSSV17, RW19, YWZ20], our protocol uses the concept of an n-party *authenticated bit*, where each party's share is authenticated by every other party with a MAC. Because of its nice algebraic properties, we choose the information theoretic MAC scheme where $\texttt{Sign}_\Delta(x)$ samples $K[x] \leftarrow \mathbb{F}_{2^\kappa}$ and outputs $(K[x], M^i[x] := K[x] + x\Delta)$. The verification procedure checks that $M^i[x] \overset{?}{=} K[x]+x\Delta$. Each party's share of an authenticated bit $\langle x \rangle^i = (x^i, \{M_j^i[x^i]\}_{j \neq i}, \{K^i[x^j]\}_{j \neq i})$

is its share of the bit x^i, the MACs of its bit for each other party $M_j^i[x^i]$, and the randomness for the MAC of each other party's bit $K^i[x^j]$.

Definition 2.2. Authenticated bit

An n-party *authenticated bit* $\langle b \rangle$ is a linearly-homomorphic authenticated sharing scheme consisting of a tuple of algorithms ($\texttt{KeyGen}, \texttt{Share}, \texttt{Verify}$)

- $\texttt{KeyGen}(n, \kappa)$: This is a randomized algorithm that takes as input the number of parties and the security parameter, and outputs key $\Delta^1, \ldots, \Delta^n \in \mathbb{F}_{2^\kappa}$
- $\texttt{Share}_{\boldsymbol{\Delta}}(b^1, \ldots, b^n)$: This is a randomized algorithm parameterized by a set of keys and takes as input additive shares of a bit b, and outputs an authenticated bit $\langle b \rangle$
- $\texttt{Verify}_{\boldsymbol{\Delta}}(\langle b \rangle)$: This algorithm is parameterized by a set of keys and takes as input an authenticated bit, and if $\langle b \rangle$ is valid under $\boldsymbol{\Delta}$ outputs b, otherwise outputs $\perp$.

Additionally, an authenticated bit satisfies the following security property: Let A and B be the distributions created by any $n-1$ shares of $\langle 0 \rangle$ and $\langle 1 \rangle$ respectively. Then, $A = B$ are identically distributed.

As authenticated bits are linearly homomorphic, $\langle x + y \rangle = \langle x \rangle + \langle y \rangle$ and $\langle cx \rangle = c\langle x \rangle$. By distributing addition and constant multiplication through the components of each share, we have e.g. $\langle x + y \rangle^i = ([x]^i + [y]^i, \{M_j^i[[x]^i] + M_j^i[[y]^i]\}, \{K^i[[x]^j] + K^i[[y]^j]\})$. By default $\langle x \rangle$ refers to an authenticated bit amongst all n parties P, and we will use the notation $\langle x \rangle^S$ to denote an m-party authenticated bit shared by some $S \subseteq P$.

Definition 2.3. Threshold authenticated bit

A "t-of-n threshold authenticated bit" $\langle\langle b \rangle\rangle$ is an *authenticated bit* scheme ($\texttt{KeyGen}, \texttt{Share}, \texttt{Verify}$) with an additional conversion procedure:

- $\texttt{Conv}^S(\cdot)$: Any $m \geq t$ sized set of parties S can (locally) convert their threshold authenticated bit into a valid m-party authenticated bit $\langle b \rangle^S = \texttt{Conv}^S(\langle\langle b \rangle\rangle)$ representing the same value.

Threshold authenticated bits satisfy an analogous security property to authenticated bits, except that the distributions are with respect to $t-1$ shares instead of $n-1$ shares. Let A and B be the distributions created by any $t-1$ shares of $\langle\langle 0 \rangle\rangle$ and $\langle\langle 1 \rangle\rangle$ respectively. Then, $A = B$ are identically distributed.

Like authenticated bits, threshold authenticated bits are also linearly homomorphic by simply distributing the operations to each of the components.

Definition 2.4. Packed threshold authenticated bits

A *packed t-of-n threshold authenticated bit* $\langle\langle x \rangle\rangle$ is an authenticated secret sharing scheme $(\texttt{KeyGen}, \texttt{Share})$ of elements $x \in \mathbb{F}_{2^\kappa}$ with a conversion procedure

- $\texttt{Conv}^S(\cdot)$: Any $m \geq t$ sized set of parties S can (locally) convert their packed threshold authenticated bits into κ valid m-party authenticated bits $(\langle b_1 \rangle^S, \ldots, \langle b_\kappa \rangle^S) = \texttt{Conv}^S(\langle\langle x \rangle\rangle)$ such that $b_1 = \texttt{Bits}(x)_1, \ldots, b_\kappa = \texttt{Bits}(x)_\kappa$.

For packed authenticated shares we equivocate between the field element representation $x \in \mathbb{F}_{2^\kappa}, \langle\langle x \rangle\rangle$ and the bitwise representation $\boldsymbol{b} = \texttt{Bits}(x) \in \{0,1\}^\kappa, \langle\langle \boldsymbol{b} \rangle\rangle$.

Instead of embedding a single bit as a field element in $\mathbb{F}_{2^\kappa}$, an arbitrary element $x \in \mathbb{F}_{2^\kappa}$ is secret shared. To convert into the individual authenticated bits for $k \in [\kappa]$, the k^{th} authenticated bit $\langle b_k \rangle^i$ is extracted using the k^{th} bit of the interpolation $\texttt{Bits}(\lambda_i^S(0) \cdot [\![x]\!]^i)_k$, by applying the Shamir reconstruction algorithm. The corresponding keys and MACs are calculated by lifting the function $\texttt{Lift}(\lambda_i^S(0) \cdot [\![\cdot]\!]^i) : \mathbb{F}_{2^\kappa}^\kappa \to \mathbb{F}_{2^\kappa}^\kappa$ that applies the bit operations of $\lambda_i^S(0) \cdot [\![\cdot]\!]^i$ to the κ values $K^i[\boldsymbol{b}^j]$ and $M_j^i[\boldsymbol{b}^i]$ respectively. To pack $\ell > \kappa$ bits, the ℓ can be partitioned into $\beta = \lceil \ell / \kappa \rceil$ blocks of size κ and authenticated blockwise. Thus, a vector of bits $\boldsymbol{b} \in \{0,1\}^\ell$ can be pack-shared as β shares $\langle\langle \boldsymbol{b} \rangle\rangle = (\langle\langle x_1 \rangle\rangle, \ldots, \langle\langle x_\beta \rangle\rangle)$.

2.2 Universal Composability

We use the Simple UC [CCL15] variant of the UC model [Can20] to analyze security. In this model, an environment $\mathcal{Z}$ that is attempting to run a protocol cannot distinguish whether it is running with the real protocol π and an adversary $\mathcal{A}$, or with a simulator $\mathcal{S}$ with access to the ideal functionality $\mathcal{F}$. This environment is given complete control over the adversary $\mathcal{A}$ as well as provides and sees the inputs and outputs for *every* party $\mathcal{P}^i$. Additionally the environment receives the non-uniform advice string z to model the composition of π with another protocol.

Definition 2.5. SUC-securely computes, [CCL15, Definition 2.2]

Let π be a protocol for up to m parties and let $\mathcal{F}$ be an ideal functionality. We say that π SUC-securely computes $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for every PPT environment $\mathcal{Z}$ there exists a negligible function $\mu(\cdot)$ such that for every n and poly-n-sized advice string z

$$|\Pr[\texttt{SUC-REAL}_{\pi,\mathcal{A},\mathcal{Z}}(n,z) = 1] - \Pr[\texttt{SUC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z) = 1]| \leq \mu(n)$$

In this strong model, the main composition theorem states that if π SUC-securely computes $\mathcal{F}$ (in the $\mathcal{H}$-hybrid model), then for a SUC-secure protocol ρ in the $\mathcal{F}$-hybrid model the composed protocol $\pi \circ \rho$ is secure in the Real (resp. $\mathcal{H}$-hybrid) model.

The SUC model differs from the UC model most prominently through the assumption of authenticated communication between parties and explicit adversarial control over message delivery. This helps to simplify the definitions of functionalities and simulators by encoding common assumptions. Additionally there is an explicit "Functionality" $\mathcal{F}$ party that the parties communicate with directly compared to the modelling of $\mathcal{F}$ as a subroutine in UC. Whereas the contents of a message from $\mathcal{P}^i$ to $\mathcal{P}^j$ can be observed by the adversary, a message from $\mathcal{P}^i$ to $\mathcal{F}$ consists of a public and private component and the adversary may only view the public component. The public component of messages to and from functionalities is the "command", session id, and other shared information between parties such as the description of a circuit. The data component of messages, such as shares of values or commitments, are private.

In this work, we prove *security with abort*. That is, the adversary is allowed to instruct the functionality to abort at any time, including after it learns the output but before the honest parties learn the output. We assume that all functionalities are equipped with an *abort* instruction implicitly, but additionally write explicitly this interaction as delivering the adversary's output to $\mathcal{S}$ and asking if it should abort or deliver the output to all parties. This is to help clarify when a cheat is expected to occur in the protocol, compared to other extraneous aborts.

3 Functionalities

One important detail of UC security that will be used in each of the functionalities are session ids or sids. sids are used to distinguish between instances of functionality calls and to relate messages together. When we refer to a "fresh" sid we mean that it is a new agreed-upon identifier that has not previously been sent to the functionality. Further, as seen in $\mathcal{F}_{\mathsf{thresh}}$, there will be an additional sub-session id that we call eid (or eval id) to distinguish between multiple circuit evaluations using the same shared preprocessing under sid.

We formulate the reactive functionality $\mathcal{F}_{\mathsf{thresh}}$ in two stages: Setup and Eval. In the Setup stage, all parties P jointly agree upon a circuit $\mathcal{C}$ that generates an initial state and each $\mathcal{P}^i$ has some private input $\boldsymbol{x}^i \in \{0,1\}^*$. The output of the circuit $\boldsymbol{\sigma} = \mathcal{C}(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n)$ is stored internally to the functionality to be used in the Eval stage. Once in the Eval stage, repeatedly, any appropriately sized group of parties $S \subseteq P$ may interact with the functionality to evaluate a new circuit $\mathcal{C}'$ that takes as implicit input $\boldsymbol{\sigma}$ in addition to new per-party inputs $\boldsymbol{v}_i$. The output of $\mathcal{C}'$ may assign outputs to parties arbitrarily.

Note that this formulation encompasses any (secret) preprocessing model. In the Setup stage, the parties can evaluate an arbitrary circuit to produce the shared secret state, and then in the Eval stage the functionality ensures that this state is correctly applied. One simple example is coin tossing, where each party provides a bit string $\boldsymbol{x}^i \in \{0,1\}^m$ and the circuit $\mathcal{C}$ simply XORs together each input. A more complicated example is to sample an AES-128 key and expand the AES key schedule. Like coin tossing, this setup circuit $\mathcal{C}$ first combines each party's random input to create the shared key, and then evaluates the key schedule which involves iteratively applying the non-linear AES S-Box to derive the full 1408-bit key schedule. In the Eval stage, a set of parties S may wish to encrypt a message under the shared key, and thus would provide the circuit $\mathcal{C}'$ that computes $\texttt{AES-128}_k(m)$ where k is the preprocessing state, and m is either hardcoded and public or derived from the inputs of each of the parties in S. Not only does $\mathcal{F}_{\mathsf{thresh}}$ ensure that for multiple executions and multiple sets of parties the input $\boldsymbol{\sigma}$ is used consistently, it provides an online advantage by reducing the size of $\mathcal{C}'$ by the size of the shared preprocessing circuit $\mathcal{C}$.

Functionality 3.1. $\mathcal{F}_{\mathsf{thresh}}(\kappa, t, n)$

This reactive functionality is parameterized by the security parameter κ, a threshold t, and the number of parties n. It interacts with parties $\mathcal{P}^1, ..., \mathcal{P}^n$ in a set P and an ideal adversary $\mathcal{S}$.

Setup: Upon receiving $(\texttt{setup}, \mathsf{sid}, \mathcal{C}, \boldsymbol{x}^i)$ from all parties $\mathcal{P}^i$, where sid is the session id, $\boldsymbol{x}^i \in \{0,1\}^{I_i}$, and $\mathcal{C} : (\{0,1\}^{I_1} \times \cdots \times \{0,1\}^{I_n}) \to \{0,1\}^O$ is the public boolean circuit which takes input from each party and produces a secret output of size O, compute $\sigma = \mathcal{C}(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n)$ and record $(\mathsf{sid}, \texttt{ready}, \sigma)$. Send $(\texttt{continue?}, \mathsf{sid})$ to $\mathcal{S}$. If they respond with $(\texttt{continue}, \mathsf{sid})$, send the confirmation $(\texttt{init-complete}, \mathsf{sid})$ to all $\mathcal{P}^i$. Otherwise send $(\texttt{abort}, \mathsf{sid})$ to each $\mathcal{P}^i$.

Eval: Let $S \subseteq P$ with $m = |S| \geq t$ be a set of parties. Let eid be the evaluation id.

Upon receiving $(\texttt{eval}, \mathsf{sid}||\mathsf{eid}, S, \mathcal{C}, \boldsymbol{v}^i)$ from all parties $\mathcal{P}^i$ for $i \in S$, where $\boldsymbol{v}^i \in \{0,1\}^{I_i}$ and $\mathcal{C} : (\{0,1\}^{|\sigma|} \times \{0,1\}^{I_i} \times \cdots \times \{0,1\}^{I_m}) \to (\{0,1\}^{O_1} \times \cdots \{0,1\}^{O_m})$ is the public circuit which takes as input the secret state $\boldsymbol{\sigma}$ and inputs from each party $i \in S$ of size I_i then produces an output to each party $\mathcal{P}^i$ of size O_i, if a record of the form $(\mathsf{sid}, \texttt{ready}, \sigma)$ exists and eid is fresh, compute $(\boldsymbol{y}^1, \ldots \boldsymbol{y}^m) = \mathcal{C}(\sigma; \boldsymbol{v}^1, \ldots, \boldsymbol{v}^m)$.

Send $(\texttt{continue?}, \mathsf{sid}||\mathsf{eid}, \{\boldsymbol{y}^j\}_{j \in \mathcal{B}})$ to $\mathcal{S}$. If they respond with the message $(\texttt{continue}, \mathsf{sid}||\mathsf{eid})$, send $(\texttt{output}, \mathsf{sid}||\mathsf{eid}, \boldsymbol{y}^i)$ to each $\mathcal{P}^i$. Otherwise send $(\texttt{abort}, \mathsf{sid}||\mathsf{eid})$ to each $\mathcal{P}^i$.

3.1 Building Blocks

Standard Functionalities The parties make use of standard coin tossing and zero-testing functionalities, $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{is-zero}}$ respectively. These both can be realized by simple folkloric methods. $\mathcal{F}_{\mathsf{rand}}$ is realized in the Random Oracle model by having each party commit to a seed, decommit the value, and use the sum of each parties' seeds together with a counter as input to the Random Oracle to receive the shared random value. $\mathcal{F}_{\mathsf{is-zero}}$ can be realized in the $\mathcal{F}_{\mathsf{mult}}$

hybrid model by having the parties check if the input $[x]$ multiplied with a uniform secret field element r is 0. Naturally, except with negligible probability that $r = 0$, if the output is 0 then x is zero as well, and any non-zero element x will be mapped to a uniform field element. $\mathcal{F}_{\mathsf{mult}}$ itself can be realized by the multiplication protocol of [DKLs19] in the correlated OT hybrid model.

Functionality 3.2. $\mathcal{F}_{\mathsf{rand}}(n, \mathcal{X})$. Coin Tossing

This functionality is parameterized by the number of parties n and any efficiently samplable distribution $\mathcal{X}$.
Sample: Upon receiving $(\texttt{sample}, \mathsf{sid})$ from all $\mathcal{P}^i$ for $i \in P$ where sid is fresh, sample a value $x \leftarrow \mathcal{X}$ and send $(\texttt{value}, \mathsf{sid}, x)$ to each $\mathcal{P}^i$.

Functionality 3.3. $\mathcal{F}_{\mathsf{is-zero}}(n, \mathbb{G})$. Zero Testing

This functionality is parameterized by the number parties n and a group $\mathbb{G}$.
Test: Upon receiving $(\texttt{test}, \mathsf{sid}, x^i)$ from all $\mathcal{P}^i$ for $i \in P$ where sid is fresh and each $x^i \in \mathbb{G}$, calculate $x = \sum_i x^i$. If $x \overset{?}{=} 0_{\mathbb{G}}$ then send $(\texttt{result}, \mathsf{sid}, 1)$, otherwise send $(\texttt{result}, \mathsf{sid}, 0)$.

Functionality 3.4. $\mathcal{F}_{\mathsf{mult}}(n, \mathbb{R})$. Multiplication

This functionality is parameterized by the number parties n and a ring $\mathbb{R}$.
Multiply: Upon receiving $(\texttt{mult}, \mathsf{sid}, x^i, y^i)$ from all $\mathcal{P}^i$ for $i \in P$ where sid is fresh and each $x^i, y^i \in \mathbb{R}$, calculate $z = \sum_i x^i \cdot \sum_i y^i$. For $i \in [n-1]$ uniform elements $z^i \leftarrow \mathbb{R}$, and set $z^n = z - \sum_{i \in [n-1]} z^i$. Send $(\texttt{mult}, \mathsf{sid}, z^i)$ to each $\mathcal{P}^i$.

Authenticated MPC We first give the functionality $\mathcal{F}_{\mathsf{aBit}}$ that creates authenticated bits $\langle \cdot \rangle$ according to the authenticated bit scheme Def. 2.2. This allows the parties to store global keys $\boldsymbol{\Delta}$ and create batches of authenticated bits $\langle \boldsymbol{x} \rangle$ under said keys $\boldsymbol{\Delta}$. $\mathcal{F}_{\mathsf{aBit}}$ can be realized in the correlated OT extension hybrid model using the protocols within [WRK17, HSSV17, YWZ20].

Functionality 3.5. $\mathcal{F}_{\mathsf{aBit}}(\kappa, \langle \cdot \rangle, n)$, Authenticated Bits

This functionality is parameterized by the security parameter κ and runs with a group of parties P of size n and an ideal adversary $\mathcal{S}$. It creates authenticated bits for the scheme $\langle \cdot \rangle = (\texttt{Share}_{\boldsymbol{\Delta}}(\cdot), \texttt{Verify}_{\boldsymbol{\Delta}}(\cdot))$.
Init: Upon receiving $(\texttt{init}, \mathsf{sid}, \Delta^i)$ from each $\mathcal{P}^i$ with $\Delta^i \in \mathbb{F}_{2^\kappa}$, store $(\mathsf{sid}, \texttt{ready}, \boldsymbol{\Delta})$ and send $(\mathsf{sid}, \texttt{ready})$ where $\boldsymbol{\Delta} = (\Delta^1, ..., \Delta^n)$.
Authentication: Upon receiving $(\texttt{aBit}, \mathsf{sid}, \ell, \boldsymbol{x}^i)$ from each $\mathcal{P}^i$ where $\boldsymbol{x}^i \in \{0,1\}^\ell$, and if a record of the form $(\mathsf{sid}, \texttt{ready}, \boldsymbol{\Delta})$ exists, then

1. For each corrupt party $\mathcal{P}^j \in \mathcal{B}$, receive $(\texttt{aBit}, \mathsf{sid}, \langle \boldsymbol{x} \rangle^j)$ from $\mathcal{S}$.

2. Sample ℓ authenticated bits $\langle \boldsymbol{x} \rangle \leftarrow \texttt{Share}_{\boldsymbol{\Delta}}(\boldsymbol{x}^1, \ldots \boldsymbol{x}^n)$ conditioned on the adversarially chosen values $\{\langle \boldsymbol{x} \rangle^j\}_{j \in \mathcal{B}}$.

Send $(\texttt{aBit}, \mathsf{sid}, \langle \boldsymbol{x} \rangle^i)$ to each $\mathcal{P}^i$.

Lastly we give the reactive MPC functionality $\mathcal{F}_{\mathsf{MPC}}$ that is parameterized by and works with an authenticated bit scheme $\langle \cdot \rangle$. Once initialized with global keys $\boldsymbol{\Delta}$, $\mathcal{F}_{\mathsf{MPC}}$ allows parties to provide and evaluate multiple boolean circuits $\mathcal{C}_1, \ldots, \mathcal{C}_m$ reactively. These circuits $\mathcal{C}$ take an input from each party $\mathcal{P}^i$ of size I_i and give output of size O_i to $\mathcal{P}^i$. Additionally $\mathcal{C}$ may contain special input and output wires for authenticated inputs $\langle \boldsymbol{\sigma} \rangle$ and outputs $\langle \boldsymbol{\sigma'} \rangle$.

This is in contrast to a typical GMW-style functionality where the functionality maintains a state of secret-shared values and allows the protocol to evaluate or open expressions using the ids of the secret state. Explicitly modeling $\mathcal{F}_{\mathsf{MPC}}$ as receiving and providing secret shares $\langle \cdot \rangle$ constrains the list of protocols that realize it, but allows computation using $\langle \cdot \rangle$ of unknown provenance such as our application. Like $\mathcal{F}_{\mathsf{aBit}}$ the protocols of [WRK17, HSSV17, YWZ20] (implicitly) realize $\mathcal{F}_{\mathsf{MPC}}$ with small modifications to allow authenticated inputs and outputs.

While the above papers all construct multiparty garbling schemes, our work does not depend on the implementation details of garbled circuits specifically and another implementation of $\mathcal{F}_{\mathsf{MPC}}$ would suffice. In the two-party case the OT-based protocol of [NNOB12] constructs an analogous functionality, and indeed as a building block [HSSV17] creates an n-party variant of [NNOB12] to produce authenticated AND triples for its garbling construction.

Functionality 3.6. $\mathcal{F}_{\mathsf{MPC}}(\kappa, \langle \cdot \rangle, n)$**. Multiparty Computation**

This functionality is parameterized by the security parameter κ, an authenticated bit scheme $\langle \cdot \rangle = (\texttt{ABit.Share}_{\Delta}(\cdot), \texttt{ABit.Verify}_{\Delta}(\cdot))$, runs with a group of parties P of size n and an ideal adversary $\mathcal{S}$.

Init: Upon receiving $(\texttt{init}, \mathsf{sid}, P, \Delta^i)$ from all $\mathcal{P}^i \in P$ with sid fresh and each $\Delta^i \in \mathbb{F}_{2^\kappa}$, record $(\mathsf{sid}, \texttt{ready}, P, \boldsymbol{\Delta})$ with $\boldsymbol{\Delta} = (\Delta^1, \ldots, \Delta^n)$ to memory and send $(\texttt{ready}, \mathsf{sid})$ to each $\mathcal{P}^i$.

Eval: Upon receiving $(\texttt{eval}, \mathsf{sid}, P, \mathcal{C}, \langle \boldsymbol{\sigma} \rangle^i, \boldsymbol{x}^i)$ from all parties $\mathcal{P}^i \in P$, with the agreed-upon circuit $\mathcal{C} : (\{0,1\}^{|\boldsymbol{\sigma}|} \times \{0,1\}^{I_1} \times \cdots \{0,1\}^{I_n}) \to (\{0,1\}^{|\boldsymbol{\sigma'}|} \times \{0,1\}^{O_1} \times \cdots \{0,1\}^{O_n})$ and each $\boldsymbol{x}^i \in \{0,1\}^{I_i}$, if there exists a record in memory of the form $(\mathsf{sid}, \texttt{ready}, P, \boldsymbol{\Delta})$, and $\texttt{ABit.Verify}_{\boldsymbol{\Delta}}(\langle \boldsymbol{\sigma} \rangle) \neq \perp$, then compute $(\boldsymbol{\sigma'}, \boldsymbol{y}^1, \ldots \boldsymbol{y}^n) = \mathcal{C}(\boldsymbol{\sigma}; \boldsymbol{x}^1, ..., \boldsymbol{x}^n)$. Otherwise send $(\texttt{abort}, \mathsf{sid})$ to all parties.

Run $\langle \boldsymbol{\sigma'} \rangle \leftarrow \texttt{ABit.Share}_{\boldsymbol{\Delta}}(\boldsymbol{\sigma'})$ and send $(\texttt{continue?}, \mathsf{sid}, \{\langle \boldsymbol{\sigma'} \rangle^j, \boldsymbol{y}^j\}_{j \in \mathcal{B}})$ to $\mathcal{S}$. If $\mathcal{S}$ responds with $(\texttt{continue}, \mathsf{sid})$ then send $(\texttt{output}, \mathsf{sid}, \langle \boldsymbol{\sigma'} \rangle^i, \boldsymbol{y}^i)$ to each $\mathcal{P}^i$. Otherwise send $(\texttt{abort}, \mathsf{sid})$ to all parties.

4 The π_{thresh} Protocol

Our protocol π_{thresh} that realizes $\mathcal{F}_{\mathsf{thresh}}$ is in the $(\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{is-zero}})$-hybrid model. All parties first sample uniform $\langle\langle \boldsymbol{r} \rangle\rangle$ which is done with the `ThreshAbits` procedure. The parties instruct $\mathcal{F}_{\mathsf{MPC}}$ with authenticated input $\langle \boldsymbol{r} \rangle$ to evaluate the circuit $\mathcal{C}'(\boldsymbol{r}, \boldsymbol{x}) = \mathcal{C}(\boldsymbol{x}) \oplus \boldsymbol{r}$ to produce the masked output $\boldsymbol{\sigma} + \boldsymbol{r}$. Once we have the masked output each party can locally calculate their threshold share $\langle\langle \boldsymbol{\sigma} \rangle\rangle = (\boldsymbol{\sigma} + \boldsymbol{r}) + \langle\langle \boldsymbol{r} \rangle\rangle$. Evaluation with a set of parties S of $\mathcal{C}_2$ simply consists of locally converting the $\langle\langle \boldsymbol{\sigma} \rangle\rangle$ into t-party authenticated bits $\langle \boldsymbol{\sigma} \rangle^S$ and then with a new t-party instance of $\mathcal{F}_{\mathsf{MPC}}$ evaluate $\mathcal{C}_2$ on $(\langle \boldsymbol{\sigma} \rangle^S, \boldsymbol{x})$

We give the protocol in two parts, first the main protocol that runs the whole experiment and ties together $\mathcal{F}_{\mathsf{MPC}}$, and second is the sub-procedure `ThreshAbits` that samples the threshold authenticated bits $\langle\langle \boldsymbol{r} \rangle\rangle$. The latter is given in more detail in the next subsection.

Protocol 4.1. $\pi_{\mathsf{thresh}}(\kappa, n, t)$**. Threshold MPC**

This protocol is parameterized by the security parameter κ, number of parties n, and the threshold t. It runs amongst a group of parties P and is defined in the $(\mathcal{F}_{\mathsf{MPC}}, \mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{is-zero}})$-hybrid model.

Setup: Upon receiving $(\texttt{setup}, \mathsf{sid}, \mathcal{C}, \boldsymbol{x}^i)$ as input from the environment $\mathcal{Z}$, with $\mathcal{C} : (\{0,1\}^{I_1} \times \cdots \{0,1\}^{I_n}) \to \{0,1\}^O$, each party $\mathcal{P}^i \in P$ does the following:

1. Sample a value $\Delta^i \leftarrow \mathbb{F}_{2^\kappa}$.
2. Sends $(\texttt{init}, \mathsf{sid}, P, \Delta^i)$ to $\mathcal{F}_{\mathsf{MPC}}$ and receives $(\texttt{ready}, \mathsf{sid})$.
3. Sends $(\texttt{init}, \mathsf{sid}, \Delta^i)$ to $\mathcal{F}_{\mathsf{aBit}}$ and receives $(\texttt{ready}, \mathsf{sid})$.
4. Run $\langle\langle \boldsymbol{r} \rangle\rangle^i \leftarrow \texttt{ThreshAbits}(\kappa, P, t, O, \Delta^i)$ to receive O packed random bits.
5. Let $\mathcal{C}'$ be the circuit that takes $(\boldsymbol{r}, \boldsymbol{x}^1, \ldots \boldsymbol{x}^n)$ as input and outputs $\mathcal{C}(\boldsymbol{x}^1, \ldots \boldsymbol{x}^n) \oplus \boldsymbol{r}$ to each party.

   Sends $(\texttt{eval}, \mathsf{sid}, P, \mathcal{C}', \langle r \rangle^i = \texttt{Conv}^P(\langle\langle \boldsymbol{r} \rangle\rangle^i), \boldsymbol{x}^i)$ to $\mathcal{F}_{\mathsf{MPC}}$. If $\mathcal{F}_{\mathsf{MPC}}$ responds with $(\texttt{abort}, \mathsf{sid})$ then output $(\texttt{abort}, \mathsf{sid})$ to $\mathcal{Z}$. Otherwise receive O masked bits of the state $(\texttt{output}, \mathsf{sid}, \boldsymbol{\sigma} + \boldsymbol{r})$ where $\boldsymbol{\sigma} = \mathcal{C}(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n)$.
6. For each chunk $a \in [\lceil O/\kappa \rceil]$ each party now defines its packed threshold resharing $\langle\langle \rho_a \rangle\rangle^i := \texttt{Combine}(\{\sigma_k + r_k\}_{k \in [(a-1)\kappa, a\kappa]}) + \langle\langle \boldsymbol{r}_{[(a-1)\kappa, a\kappa]} \rangle\rangle^i$ where $\langle\langle \boldsymbol{r}_{[(a-1)\kappa, a\kappa]} \rangle\rangle^i$ is the a^{th} packed share.
7. Records its share of the threshold authenticated bits $(\mathsf{sid}, \langle\langle \boldsymbol{\sigma} \rangle\rangle^i = \langle\langle \boldsymbol{\rho} \rangle\rangle^i)$ using the packed authenticated bits $\langle\langle \boldsymbol{\rho} \rangle\rangle^i$.

Eval: Let S be the set of parties wishing to run the evaluation with $m = |S| \geq t$. Let $\texttt{Conv}^{i,S}(\cdot)$ be the function that converts the packed threshold authenticated bits into authenticated bits, according to Definition 2.4.

Upon receiving $(\texttt{eval}, \mathsf{sid}, \mathsf{eid}, S, \mathcal{C}_2, \boldsymbol{v}^i)$ as input from $\mathcal{Z}$ where $\mathcal{C}_2 : (\{0,1\}^{|\sigma|} \times \{0,1\}^{I_1} \times \cdots \{0,1\}^{I_m}) \to (\{0,1\}^{O_1} \times \cdots \{0,1\}^{O_m})$. If $i \in S$, eid is fresh, and a record of the form $(\mathsf{sid}, \langle\langle \boldsymbol{\sigma} \rangle\rangle^i)$ exists, each party $\mathcal{P}^i$ does the following:

1. Calculates its share of the m-party authenticated state, $\langle \boldsymbol{\sigma} \rangle^i = \texttt{Conv}^{i,S}(\langle\langle \boldsymbol{\sigma} \rangle\rangle^i)$:
2. Sends $(\texttt{init}, \mathsf{sid}||\mathsf{eid}, S, \Delta^i)$ to $\mathcal{F}_{\mathsf{MPC}}$, and receives $(\texttt{ready}, \mathsf{sid})$
3. Sends $(\texttt{eval}, \mathsf{sid}||\mathsf{eid}, S, \mathcal{C}_2, \langle \boldsymbol{\sigma} \rangle^i, \boldsymbol{v}^i)$ to $\mathcal{F}_{\mathsf{MPC}}$.

   If $\mathcal{F}_{\mathsf{MPC}}$ responds with $(\texttt{abort}, \mathsf{sid}||\mathsf{eid})$ then output $(\texttt{abort}, \mathsf{sid}||\mathsf{eid})$ to $\mathcal{Z}$. Otherwise receive the circuit output $(\texttt{output}, \mathsf{sid}||\mathsf{eid}, \boldsymbol{y}^i)$.
4. Output $(\texttt{output}, \mathsf{sid}, \mathsf{eid}, \boldsymbol{y}^i)$ to $\mathcal{Z}$.

4.1 The ThreshAbits Procedure

In this section, we describe the procedure for creating packed threshold authenticated bits (Def. 2.4).

Consider the scenario when the parties wish to sample a single random element x. First, each party $\mathcal{P}^i$ holds a value $x^i \in \mathbb{F}_{2^\kappa}$, and we define $x = \sum_i x^i$. The parties each distribute Shamir shares of their value x^i by sending $[\![x^i]\!]^j \in \mathbb{F}_{2^\kappa}$ to each other party $\mathcal{P}^j$. From this the parties can construct the threshold share of x, $[\![x]\!]^i = \sum_j [\![x^j]\!]^i$. We do want our output to have the structure of a threshold authenticated bit, so the parties jointly calculate authenticated bits $\langle [\![x]\!] \rangle$ using their individual shares $[\![x]\!]^i$ as input to $\mathcal{F}_{\mathsf{aBit}}$. Up until this point, the steps are folklore for secret sharing a random bit without a malicious adversary.

A malicious adversary can send values $[\![x^j]\!]^i$ to each honest $\mathcal{P}^i$ that are not of the correct degree. In order to verify that secret shares $[\![x]\!]$ are of the correct degree, or that the polynomial defined by $\forall i \in P. p(i) = [\![x]\!]^i$ is of degree $t-1$, define the global predicate f that takes as input each of the parties' threshold shares, $([\![x]\!]^1, \ldots, [\![x]\!]^n)$ and outputs 0 if the interpolation of $\{(i, [\![x]\!]^i)\}_{i \in P}$ is a degree $t-1$ polynomial. Instead of fully reconstructing the polynomial p, this predicate can be efficiently implemented with a randomized check using the Schwartz-Zippel lemma and properties of the interpolation polynomial.

Lemma 4.2 (Schwartz-Zippel Polynomial Identity Testing [DKSS13]). *Let $p \in \mathbb{F}_{2^\kappa}[X]$ be a non-zero polynomial of degree d, then*

$$\Pr_{r \in \mathbb{F}_{2^\kappa}}[p(r) = 0] \leq \frac{d}{|\mathbb{F}_{2^\kappa}|}$$

The polynomial p that represents the secret shares will not be the 0 polynomial with high probability. Then we need to construct a polynomial p' that is 0 iff p is of degree $\leq t-1$. Fortunately it is easy to construct a polynomial of degree at most $t-1$ by interpolating over a set S of size t, $\{i, x_i\}_{i \in S}$, and moreover this polynomial is unique. If you take a polynomial g of degree $t-1$ and any two distinct set of points S_1, S_2 of size $\geq t$ then the interpolated polynomials of $\{i, g(i)\}_{i \in S_1}$ and $\{j, g(j)\}_{j \in S_2}$ are identical. We want to rely on, and show, that the inverse of this statement that $\deg(g) > t-1$ implies that the two interpolations are different.

Lemma 4.3 (Uniqueness of Interpolation Polynomials)**.** *Let $p \in \mathbb{F}_{2^\kappa}[X]$ be a non-zero polynomial of degree $t \leq d < 2t$ and let $S_1, S_2 \subset \mathbb{F}_{2^\kappa}$ be two sets of points where $|S_1| = |S_2| = t$ and $|S_1 \cup S_2| > d$. Let p_1 and p_2 be the degree $t-1$ polynomial interpolation of $\{i, p(i)\}_{i \in S_1}$ and $\{j, p(j)\}_{j \in S_2}$ respectively, then $p_1 \neq p_2$.*

Proof. Suppose that $p_1 = p_2$, then for all $i \in S_1 \cup S_2 . p_1(i) = p_2(i)$. Let f be the polynomial interpolation of $\{i, p_1(i)\}_{i \in S_1 \cup S_2}$. By definition $f = p_1 = p_2$ and is of degree $t-1$. But the polynomial p of degree $d > t-1$ is also interpolated by the points $\{i, p_1(i)\}_{i \in S_1 \cup S_2}$ in contradiction to the assumption that the degree of p is greater than $t-1$. $\square$

If the threshold secret share $[\![x]\!]$ represents a polynomial p of degree t' greater than $t-1$, then reconstructing with two subsets of points of size $t-1$ will yield two different polynomials. By the Schwartz-Zippel lemma, we can test if a given polynomial g is the zero polynomial by evaluating g on a random point r and checking if $g(r) = 0$. Our output of our initial predicate f is simply the output of $g(r)$ that can be calculated by interpolating at r $\sum_{i \in S_1} \lambda_i^{S_1}(r)[\![x]\!]^i + \sum_{j \in S_2} \lambda_j^{S_2}(r)[\![x]\!]^j \stackrel{?}{=} 0$.

However, it is not clear how to evaluate the predicate f directly, because no one party has all of the secret shares by design. Instead, each party holds authenticated bits of shares $\langle [\![x]\!] \rangle$ and we can utilize the homomorphic properties of the authenticated bit scheme. Observe that f consists of two linear components: sums and constant multiplications, and an equality check. The linearity of the secret sharing scheme $\langle \cdot \rangle$ can be used to evaluate the linear portion of f on the authenticated bits $\langle z \rangle = f(\langle [\![x]\!] \rangle)$. In an honest execution when $[\![x]\!]$ is indeed a degree $t-1$ polynomial, then $f([\![x]\!]) = 0$ and the parties can compute $\langle z = 0 \rangle$ where each party holds z^i with $\sum_i z^i = 0$.

We wish to check that $\langle z \rangle$ is in fact a valid authenticated share of 0 without revealing the values $\boldsymbol{z}$ as that would leak the threshold shares of the honest parties. We will do this by reducing to the unforgeability of the underlying MAC scheme. Unfortunately in this way we do not identify which particular party cheated, only that a cheat does exist. If $z = 0$ then $\mathcal{P}^i$ knows exactly what the sum of each other $\mathcal{P}^j$'s bits should be, namely its own value $z^i = \sum_{j \neq i} z^j$. Party $\mathcal{P}^i$ can calculate the expected sum of the MACs of the other parties $M_i^i[z^i] := \sum_{j \neq i} K^i[z^j] + z^i \Delta^i = \sum_{j \neq i} M_i^j[z^j]$ even though it doesn't know the individual values z^j. Checking if $\sum_j M_i^j[z^j] \overset{?}{=} 0$ for each $i \in P$ individually would verify that all parties hold correct shares under $\mathcal{P}^i$'s key Δ^i. However, this requires a zero-test per party, so we instead simultaneously test all parties' inputs by checking that $\sum_i \sum_j M_j^i[z^i] \overset{?}{=} 0$ where each party $\mathcal{P}^i$ supplies the input $[z]^i = \sum_j M_j^i[z^i]$. Intuitively, if the adversary cheated in the creation of $\langle [\![x]\!] \rangle$, in order to pass this test they would need to forge some MAC $M_i^j[\tilde{z}^j] \neq M_i^j[z^j]$. But by the unforgeability of the MAC scheme the adversary can only do so with negligible probability.

Using a standard zero-testing functionality $\mathcal{F}_{\mathsf{is-zero}}$ to check $[z]$ the parties have verified that they have a valid t-of-n secret share $[\![x]\!]$. Each party can then output $\langle\langle x \rangle\rangle^i = ([\![x]\!]^i, \langle [\![x]\!] \rangle^i)$ as its share and the associated authenticated bits.

Algorithm 4.4. `ThreshAbits`$(\kappa, P, t, O, \Delta^i)$**. Packed Threshold Authenticated Bits**

This procedure is parameterized by the security parameter κ, a set of parties P, a threshold t, a number of elements to sample O, and the party's correlation Δ^i. Let $S_1, S_2 \subset P$ be two different sets of size t such that $S_1 \cup S_2 = P$. It returns the shares $\langle\langle \boldsymbol{\rho} \rangle\rangle^i$ of O random bits, or $\perp$ if a cheat was detected.

1. Define $\ell = \lceil O/\kappa \rceil$ as the number of packed shares to create.
2. Run the share sampling procedure $([\![\boldsymbol{\rho}]\!]^i, \langle \boldsymbol{B} \rangle^i) \leftarrow$ `AuthShare`(ℓ) to receive ℓ Shamir shares, and authenticated bits of the bit decomposition of each share.
3. If $t < |P|$, run the degree testing procedure $o \leftarrow$ `DegreeTest`$([\![\boldsymbol{\rho}]\!]^i, \langle \boldsymbol{B} \rangle^i)$. If the output o is not equal to 1, then output $\perp$.
4. Return $\langle\langle \boldsymbol{\rho} \rangle\rangle^i = ([\![\boldsymbol{\rho}]\!]^i, \{M_j^i[\boldsymbol{B}^i]\}_{j \neq i}, \{K^i[\boldsymbol{B}^j]\}_{j \neq i})$.

This subroutine creates and authenticates threshold shares of input bits.
`AuthShare`(ℓ) :

A1. Sample ℓ threshold t-of-n shares $[\![\boldsymbol{\rho}^i]\!]$ of random values $\boldsymbol{\rho}^i \leftarrow \mathbb{F}_{2^\kappa}^{\ell}$, and send $[\![\boldsymbol{\rho}^i]\!]^j$ to each other $\mathcal{P}^j$. $\mathcal{P}^i$ then calculate the threshold share of the state, $[\![\boldsymbol{\rho}]\!]^i = \sum_{j \in P} [\![\boldsymbol{\rho}^j]\!]^i$.

A2. Use the $\mathcal{F}_{\mathsf{aBit}}$ functionality to authenticate the bits of each parties' shares. For $a \in [\ell]$, let $\boldsymbol{B}_a^i = \mathtt{Bits}([\![\rho_a]\!]^i) \in \{0,1\}^\kappa$ be the bit decomposition of the threshold share.

Send $(\mathtt{aBit}, \mathsf{sid}, \ell\kappa, \boldsymbol{B}_1^i, \ldots, \boldsymbol{B}_\ell^i)$ to $\mathcal{F}_{\mathsf{aBit}}$.

Receive $(\mathtt{aBit}, \mathsf{sid}, \langle \boldsymbol{B}_1 \rangle^i, \ldots, \langle \boldsymbol{B}_\ell \rangle^i)$.

A3. Return $[\![\boldsymbol{\rho}]\!]^i, \langle \boldsymbol{B} \rangle^i$.

We define the DegreeTest procedure that validates that the authenticated bits $\langle \boldsymbol{B}_{a,*} \rangle$ represent a valid t-of-n threshold sharing $[\![\rho_a]\!]$ under Δ^*. Recall that for an authenticated bit $\langle b \rangle$ $\mathcal{P}^i$'s share $\langle b \rangle^i = (b^i, \{M^i_j[b^i]\}_{j \neq i}, \{K^i[b^j]\}_{j \neq i})$.

$\texttt{DegreeTest}([\![\boldsymbol{\rho}]\!]^i, \langle \boldsymbol{B} \rangle^i)$:

D1. Call $\mathcal{F}_{\mathsf{rand}}$ to receive a random point $r \leftarrow \mathbb{F}_{2^\kappa}$. Let

$$f_i^S(x) = \begin{cases} \lambda_i^S(x) & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

define the output when attempting to calculate a Lagrange interpolation polynomial outside of the basis.

D2. For each $a \in [\ell]$, let $M^i_j[[\![\rho_a]\!]^i] := \texttt{Combine}(M^i_j[\boldsymbol{B}^i_{a,*}])$, and likewise $K^i[[\![\rho_a]\!]^j] := \texttt{Combine}(K^i[\boldsymbol{B}^j_{a,*}])$.

$\mathcal{P}^i$ calculates the interpolation at r to check for polynomial equality

$$y^i_a = (f_i^{S_1}(r) + f_i^{S_2}(r)) \cdot [\![\rho_a]\!]^i$$

and then for each other $j \neq i$, calculates the MAC on its input

$$M^i_j[y^i_a] = (f_i^{S_1}(r) + f_i^{S_2}(r)) \cdot M^i_j[[\![\rho_a]\!]^i]$$

and its key for each other party's input

$$K^i[y^j_a] = (f_j^{S_1}(r) + f_j^{S_2}(r)) \cdot K^i[[\![\rho_a]\!]^j]$$

$\mathcal{P}^i$ can then define the expected MAC over all other party's inputs for batch a

$$z^i_a = (\sum_{j \neq i} K^i[y^j_a]) + y^i_a \Delta^i$$

D3. Combine the values z^i_* in the following way

Let $z^i = \sum_{a \in [\ell]} z^i_a$ and $M^i_j[y^i] = \sum_{a \in [\ell]} M^i_j[y^i_a]$ be the linear combination of each of the ℓ elements respectively. Then, let $[m]^i = z^i + \sum_{j \neq i} M^i_j[y^i] \in \mathbb{F}_{2^\kappa}$.

Each party sends $(\texttt{test}, \mathsf{sid}, [m]^i)$ to $\mathcal{F}_{\mathsf{is-zero}}$, and receives $(\texttt{result}, \mathsf{sid}, o)$.

D4. Return o.

4.2 Security of π_{thresh}

Theorem 4.5 (SUC Security of π_{thresh}). *The protocol π_{thresh} SUC-realizes $\mathcal{F}_{\mathsf{thresh}}$ in the ($\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{F}_{\mathsf{rand}}$, $\mathcal{F}_{\mathsf{is-zero}}$)-hybrid model against a malicious adversary* *that statically corrupts up to $t-1$ parties.*

Proof of Theorem 4.5. The protocol π_{thresh} correctly implements the ideal functionality $\mathcal{F}_{\mathsf{thresh}}$, and the Real experiment is statistically indistinguishable from the Ideal experiment in the given hybrid model:

$$\texttt{Real}_{\pi_{\mathsf{thresh}},\mathcal{A},\mathcal{Z}}(\kappa, z)_{z\in\{0,1\}^*} \approx \texttt{Ideal}_{\mathcal{F}_{\mathsf{thresh}},\mathcal{S}_{\mathsf{thresh}},\mathcal{Z}}(\kappa, z)_{z\in\{0,1\}^*}$$

We begin with a description of $\mathcal{S}_{\mathsf{thresh}}$ and proceed with a sequence of hybrids.

Simulator 4.6. $\mathcal{S}_{\mathsf{thresh}}(\kappa, n, t)$. Simulator for π_{thresh}

This simulator is parameterized by the security parameter κ, number of parties n, and the threshold t, and interacts with $\mathcal{A}$ as the functionalities $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{F}_{\mathsf{rand}}$, and $\mathcal{F}_{\mathsf{is-zero}}$. Let $\mathcal{H}$ denote the set of indicies of the honest parties and $\mathcal{B}$ denote the set indicies of the corrupt parties. Let $S_1, S_2 \subset P$ each be of size t and $S_1 \cup S_2 = P$.

Setup: Let $(\texttt{setup}, \mathsf{sid}, \mathcal{C}, \cdot)$ be the input from the environment, with $\mathcal{C} : \{0,1\}^I \to \{0,1\}^O$ a circuit. Let $\ell = \lceil O/\kappa \rceil$ be the number of threshold shares to create.

S1. As $\mathcal{F}_{\mathsf{MPC}}$, receive $(\texttt{init}, \mathsf{sid}, \Delta_1^j)$ from each $\mathcal{P}^j \in \mathcal{B}$, then send $(\texttt{ready}, \mathsf{sid})$ to each $\mathcal{P}^j$.

S2. As $\mathcal{F}_{\mathsf{aBit}}$, receive $(\texttt{init}, \mathsf{sid}, \Delta_2^j)$ from each $\mathcal{P}^j \in \mathcal{B}$, then send $(\texttt{ready}, \mathsf{sid})$ to each $\mathcal{P}^j$.

S3. For each honest party $\mathcal{P}^i \in \mathcal{H}$ and each $a \in [\ell]$ create threshold t-of-n shares $[\![\rho_a^i]\!]$ of a random element $\rho_a^i \in \mathbb{F}_{2^\kappa}$, and send $[\![\rho_a^i]\!]^j$ to each corrupt $\mathcal{P}^j \in \mathcal{B}$. From each corrupt party $\mathcal{P}^j \in \mathcal{B}$ receive the share for each honest $\mathcal{P}^i \in \mathcal{H}$, $[\![\rho_a^j]\!]^i$. Calculate the honest party shares as $[\![\boldsymbol{\rho}]\!]^i = \sum_{j\in P} [\![\boldsymbol{\rho}^j]\!]^i$.

S4. As $\mathcal{F}_{\mathsf{aBit}}$, receive $(\texttt{aBit}, \mathsf{sid}, \ell\kappa, \boldsymbol{B_1^j}, \ldots, \boldsymbol{B_\ell^j})$ and receive $(\texttt{aBit}, \mathsf{sid}, \langle \boldsymbol{B} \rangle^j)$ from $\mathcal{A}$ for each $\mathcal{P}^j \in \mathcal{B}$.

As $\mathcal{F}_{\mathsf{aBit}}$, send $(\texttt{aBit}, \mathsf{sid}, \langle \boldsymbol{B}_1 \rangle^j, \ldots, \langle \boldsymbol{B}_\ell \rangle^j)$ to each $\mathcal{P}^j \in \mathcal{B}$.

S5. As $\mathcal{F}_{\mathsf{rand}}$, receive $(\texttt{sample}, \mathsf{sid})$ from each corrupt party. Sample a random point $r \in \mathbb{F}_{2^\kappa}$ and send $(\texttt{value}, \mathsf{sid}, r)$ to each $\mathcal{P}^j$.

S6. As $\mathcal{F}_{\mathsf{is-zero}}$, receive $(\texttt{test}, \mathsf{sid}, [m']^j)$ from each corrupt party $\mathcal{P}^j$. Let $[m]^j$ be the expected honestly calculated share over

$$[\![\boldsymbol{\rho}]\!]^j, \langle \boldsymbol{B} \rangle^j, S_1, S_2, r$$

as described in Algorithm 4.4 Step D3. where each $[\![\rho_a]\!]^j = \texttt{Combine}(\boldsymbol{B}_{a,*}^j) \in \mathbb{F}_{2^\kappa}$ for all $a = [\ell]$.

S7. If either

$$\exists a \in [\ell] \ . \ \mathtt{deg}([\![\rho_a]\!]) \neq t-1$$
$$\sum_{j \in \mathcal{B}} [m']^j \neq \sum_{j \in \mathcal{B}} [m]^j$$

then as $\mathcal{F}_{\mathsf{is-zero}}$ send $(\mathtt{result}, \mathsf{sid}, 0)$ to each corrupt party $\mathcal{P}^j$ and send $(\mathtt{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{thresh}}$ and each party, then abort.

Otherwise as $\mathcal{F}_{\mathsf{is-zero}}$ send $(\mathtt{result}, \mathsf{sid}, 1)$ to each corrupt party $\mathcal{P}^j$.

S8. For each $j \in \mathcal{B}$ let $\langle\langle \boldsymbol{\rho} \rangle\rangle^j = ([\![\boldsymbol{\rho}]\!]^j, \langle \boldsymbol{B} \rangle^i)$.

S9. As $\mathcal{F}_{\mathsf{MPC}}$ receive $(\mathtt{eval}, \mathsf{sid}, \mathcal{C}', \langle \tilde{\boldsymbol{\rho}} \rangle^j, x^j)$ from each $\mathcal{P}^j \in \mathcal{B}$.

S10. Send $(\mathtt{setup}, \mathsf{sid}, \mathcal{C}, x^j)$ to $\mathcal{F}_{\mathsf{thresh}}$ as each $\mathcal{P}^j$. Receive $(\mathtt{continue?}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{thresh}}$.

Let $\langle \boldsymbol{\rho} \rangle^j = \mathtt{Conv}^P(\langle\langle \boldsymbol{\rho} \rangle\rangle^j)$. If there exists $\Delta_1^j \neq \Delta_2^j$ or if $\langle \boldsymbol{\rho} \rangle^j \neq \langle \tilde{\boldsymbol{\rho}} \rangle^j$ then send $(\mathtt{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{thresh}}$ and to each corrupt party $\mathcal{P}^j$.

S11. Sample random values $\boldsymbol{R} \in \{0,1\}^O$, and on behalf of the $\mathcal{F}_{\mathsf{MPC}}$ functionality, send the adversary the output $(\mathtt{continue?}, \mathsf{sid}, \boldsymbol{R})$ to $\mathcal{A}$.

If $\mathcal{A}$ responds with $(\mathtt{abort}, \mathsf{sid})$ then send $(\mathtt{abort}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{thresh}}$. Otherwise send $(\mathtt{continue}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{thresh}}$, and send $(\mathtt{output}, \mathsf{sid}, R)$ to each $\mathcal{P}^j$.

S12. For each corrupt party $\mathcal{P}^j \in \mathcal{B}$ calculate the expected threshold share $\langle\langle \boldsymbol{\sigma} \rangle\rangle^j = \boldsymbol{R} + \langle\langle \boldsymbol{\rho} \rangle\rangle^j$.

Eval: Let $(\mathtt{eval}, \mathsf{sid}, S, \mathcal{C}, \cdot)$ be the input from the environment, with the set of parties $S \subseteq P$ having $|S| \geq t$, and $\mathcal{C} : (\{0,1\}^\sigma \times \{0,1\}^I) \to \{0,1\}^O$.

E1. As $\mathcal{F}_{\mathsf{MPC}}$ receive $(\mathtt{init}, \mathsf{sid}||\mathsf{eid}, S, \Delta_3^j)$ from each corrupt $\mathcal{P}^j$ then send $(\mathtt{ready}, \mathsf{sid}_S)$ to each $\mathcal{P}^j$.

E2. As $\mathcal{F}_{\mathsf{MPC}}$ receive $(\mathtt{eval}, \mathsf{sid}||\mathsf{eid}, S, \mathcal{C}, \langle \tilde{\boldsymbol{\sigma}} \rangle^j, v^j)$ from each corrupt $\mathcal{P}^j$.

E3. Send $(\mathtt{eval}, \mathsf{sid}||\mathsf{eid}, S, v^j)$ to $\mathcal{F}_{\mathsf{thresh}}$ for each corrupt $\mathcal{P}^j$.

E4. Receive $(\mathtt{continue?}, \boldsymbol{y})$ from $\mathcal{F}_{\mathsf{thresh}}$.

Let $\langle \boldsymbol{\sigma} \rangle^j = \mathtt{Conv}^{i,S}(\langle\langle \boldsymbol{\sigma} \rangle\rangle^j)$. If $\langle \boldsymbol{\sigma} \rangle^j \neq \langle \tilde{\boldsymbol{\sigma}} \rangle^j$ or if there exists j such that $\Delta_3^j \neq \Delta_1^j$ then send $(\mathtt{abort}, \mathsf{sid}||\mathsf{eid})$ to $\mathcal{F}_{\mathsf{thresh}}$ and each $\mathcal{P}^j$ and abort.

Otherwise send $(\mathtt{continue?}, \mathsf{sid}||\mathsf{eid}, \boldsymbol{y})$ to $\mathcal{A}$.

If $\mathcal{A}$ responds with $(\texttt{continue}, \mathsf{sid}||\mathsf{eid})$ then send $(\texttt{continue}, \mathsf{sid}||\mathsf{eid})$ to $\mathcal{F}_{\mathsf{thresh}}$ and send $(\texttt{output}, \mathsf{sid}||\mathsf{eid}, y_j)$ to each corrupt party $\mathcal{P}^j$. If $\mathcal{A}$ responds with $(\texttt{abort}, \mathsf{sid}||\mathsf{eid})$ send $(\texttt{abort}, \mathsf{sid}||\mathsf{eid})$ to $\mathcal{F}_{\mathsf{thresh}}$ and each $\mathcal{P}^j$ and abort.

Hybrid $\mathcal{H}_0$. This is the real-world experiment $\texttt{Real}_{\pi_{\mathsf{thresh}},\mathcal{A},\mathcal{Z}}(\kappa, z)$.

Hybrid $\mathcal{H}_1$. In this hybrid, we define a simulator $\mathcal{S}_1$ that acts as $\mathcal{F}_{\mathsf{thresh}}$ for the honest parties. It simulates the honest parties and ideal functionalities $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{F}_{\mathsf{aBit}}$, $\mathcal{F}_{\mathsf{rand}}$, and $\mathcal{F}_{\mathsf{is-zero}}$ for the adversary by running their code, learning the honest parties inputs to the functionalities as well as their outputs and intercepting any messages sent by the corrupt parties to the honest parties and ideal functionalities. By running the code for the honest parties, $\mathcal{S}_1$ creates the keys Δ^i used for authentication, and by running the code for the hybrid functionalities the simulator can exactly construct the authenticated shares and $\mathcal{F}_{\mathsf{MPC}}$ outputs used within the protocol. These are syntactic changes so, $\mathcal{H}_1$ and $\mathcal{H}_0$ are identically distributed.

Hybrid $\mathcal{H}_2$. In this hybrid, we construct the simulator $\mathcal{S}_2$ that acts as $\mathcal{S}_1$ but implements Steps S4.-S8. in Setup of $\mathcal{S}_{\mathsf{thresh}}$ corresponding to Step 4 of π_{thresh}. This step samples random threshold authenticated bits to prevent an adversary cheat from creating incorrect threshold authenticated bits. Importantly $\mathcal{S}_2$ will always create threshold t-of-n authenticated bits $\langle\langle \boldsymbol{\rho} \rangle\rangle$ whereas $\mathcal{S}_1$ may create a threshold t'-of-n authenticated bit with $t' > t$. The event that $\mathcal{S}_2$ aborts where $\mathcal{S}_1$ does not occurs when either the evaluation at r happens to be 0, or when the adversary successfully creates a value m' that causes the is-zero test to pass. As these events occur with negligible probability in κ, the distribution of $\mathcal{H}_2$ is statistically indistinguishable from $\mathcal{H}_1$.

Proof. First we show that the probabilistic degree test outputs a false positive with negligible probability. If $[\![\rho]\!]$ is not a t-of-n secret sharing then the polynomial defined by $p(i) = [\![\rho]\!]^i$ for $i \in P$ is of degree $d \geq t$. Then by the Lemma 4.3, if we compute two interpolations using appropriate subsets $S_1, S_2 \subset P$ of points of size t the interpolated polynomials p_1, p_2 will always be different. If we evaluate both p_1, p_2 on a random point r as $\mathcal{S}_1$ does, then, by Lemma 4.2 $p_1(r) = p_2(r)$ only with probability $\leq \frac{d}{|\mathbb{F}_{2^\kappa}|}$. Therefore as $d \leq n$ and $n = \mathrm{poly}(\kappa)$ the degree test will emit a false positive with negligible probability.

Then if an adversary induces shares $[\![\rho]\!]$ that are on a polynomial of greater than degree $t-1$, it must also cheat in the provided value $[m]^j$ to cause the zero test of $\texttt{DegreeTest}$ to pass. The value $[m]$ is composed of terms $(\sum_{j\neq i} K^i[y^j]) + y^i\Delta^i + \sum_{j\neq i} M_i^j[y^j] = (\sum_i y^i)\Delta^i$. In the case that the degree test passes, then $\sum_i y^i = 0$, but if the degree test fails $\sum_i y^i = \delta_y \neq 0$. For an adversary that knows $\{M_i^j[y^j]\}_{j\in\mathcal{B}}$ this is equivalent to guessing the value $\delta_y\Delta^i$. But for uniform Δ^i and non-zero δ_y this occurs only with probability $\frac{1}{|\mathbb{F}_{2^\kappa}|}$.

By the union bound, the probability that both checks pass when the adversary provides inconsistent shares is $\leq \frac{1+n}{|\mathbb{F}_{2^\kappa}|} = \mathrm{negl}(\kappa)$ $\square$

Hybrid $\mathcal{H}_3$. In this hybrid, we construct $\mathcal{S}_3$ that implements Eval Steps E1.-E3. of $\mathcal{S}_{\mathsf{thresh}}$, and thus $\mathcal{S}_3$ no longer acts as $\mathcal{F}_{\mathsf{thresh}}$ for the honest parties during the Eval operation. When attempting to simulate the evaluation of the circuit $\mathcal{C}$, $\mathcal{S}_3$ no longer has the inputs of the honest parties v^i and so must rely on $\mathcal{F}_{\mathsf{thresh}}$ to calculate the output $\boldsymbol{y}$. It is still $\mathcal{S}_3$'s responsibility to direct $\mathcal{F}_{\mathsf{thresh}}$ to abort if the adversary cheated by supplying an inconsistent Δ^j or invalid shares of the state $\langle\tilde{\boldsymbol{\sigma}}\rangle^j$ to $\mathcal{F}_{\mathsf{MPC}}$. Luckily $\mathcal{S}_3$, like $\mathcal{S}_2$, creates the threshold authenticated bits $\langle\langle\boldsymbol{\sigma}\rangle\rangle$ during the Setup operation, $\mathcal{S}_3$ can exactly mimic the authenticated bit verification as performed in $\mathcal{F}_{\mathsf{MPC}}$. Then $\mathcal{S}_3$ can run the full $\mathtt{Abit.Verify}_{\boldsymbol{\Delta}}(\langle\tilde{\boldsymbol{\sigma}}\rangle)$ to abort in exactly the same cases that $\mathcal{S}_2$ would abort.

In the event that the adversary is able to produce authenticated bits $\langle\tilde{\boldsymbol{\sigma}}\rangle$ that are valid, $\mathcal{S}_3$ fails to calculate the value $\mathcal{C}(\tilde{\boldsymbol{\sigma}}, \boldsymbol{v})$ as it does not know the honest parties' components of $\boldsymbol{v}$; in contrast, $\mathcal{S}_2$ succeeds in calculating this circuit output in this event. Nonetheless, the security of the authenticated bit scheme implies that the probability that an adversary can forge a MAC (Def. 2.1) is equal to guessing the uniform key Δ^i of an honest party, or $\frac{1}{|\mathbb{F}_{2^\kappa}|}$. To forge authenticated bits the adversary must simultaneously forge MACs for each other party, but we simply bound this probability above by forging a MAC for a single party. Therefore the probability of this event occurring is negligible and the distributions of $\mathcal{H}_3$ and $\mathcal{H}_2$ are statistically indistinguishable.

Hybrid $\mathcal{H}_4$. In this hybrid $\mathcal{S}_4$ implements Setup Steps S11.-S12. of $\mathcal{S}_{\mathsf{thresh}}$ corresponding to Step 5 of π_{thresh} that samples a random output for $\mathcal{F}_{\mathsf{MPC}}$ instead of calculating the correct output $\boldsymbol{\sigma}+\boldsymbol{\rho}$. As the adversary only knows $t-1$ shares of the threshold authenticated bit $\langle\langle\boldsymbol{\rho}\rangle\rangle$ the value $\boldsymbol{\rho}$ is uniformly random and unknown to the adversary by the privacy of the threshold authenticated bit scheme 2.3. The value $\boldsymbol{\sigma}+\boldsymbol{\rho}$ is then also indistinguishable from uniform to the adversary.

Then $\mathcal{S}_4$ replacing the output of $\mathcal{F}_{\mathsf{MPC}}$, $\boldsymbol{\sigma}+\boldsymbol{\rho} \in \{0,1\}^O$, with the uniform value $\boldsymbol{R} \leftarrow \{0,1\}^O$ and calculating the expected adversary output $\langle\langle\boldsymbol{\sigma}\rangle\rangle^j = \boldsymbol{R} + \langle\langle\boldsymbol{\rho}\rangle\rangle^j$ is indistinguishable from the distribution of $\mathcal{H}_3$. An adversary that knows the correct circuit output $\boldsymbol{\sigma} = \mathcal{C}(\boldsymbol{x})$ would be able to calculate the expected shares $[\![\boldsymbol{\sigma}]\!]^i$ for each honest $\mathcal{P}^i$ as $\boldsymbol{\sigma}$ is a t$^{\text{th}}$ point on the shared polynomial. As the simulator does not know $\boldsymbol{\sigma}$ it would be unable to produce the "correct" values $\langle\langle\boldsymbol{\sigma}\rangle\rangle^i$ except with negligible probability However, the adversary never sees the corresponding honest party shares of the threshold authenticated bit $\langle\langle\boldsymbol{\sigma}\rangle\rangle^i$ to be able to identify that the simulator has cheated.

The distribution of $\mathcal{H}_4$ is identical to that of $\mathcal{H}_3$.

Hybrid $\mathcal{H}_5$. In this hybrid $\mathcal{S}_5$ no longer generates a key Δ^i for each honest party and likewise does not generate honest party shares of authenticated bits. $\mathcal{S}_5$ implements Setup Step S10. and Eval Step E4. of $\mathcal{S}_{\mathsf{thresh}}$ to verify if the authenticated bits from an adversary are valid. Unlike $\mathcal{S}_4$, $\mathcal{S}_5$ only checks equality of the input to $\mathcal{F}_{\mathsf{MPC}}$ with the expected values derived from $\langle\langle\boldsymbol{\rho}\rangle\rangle^j$ and $\langle\langle\boldsymbol{\sigma}\rangle\rangle^j$

instead of running the code of $\mathcal{F}_{\mathsf{MPC}}$ that evaluates the $\texttt{Abit.Verify}_{\boldsymbol{\Delta}}(\langle \boldsymbol{\sigma} \rangle)$ algorithm with honest party keys and shares. In the event that the adversary is able to forge valid authenticated bits $\langle \tilde{\boldsymbol{\sigma}} \rangle$ $\mathcal{S}_5$ would abort whereas $\mathcal{S}_4$ would not. Again, by the security of the authenticated bit scheme the probability that an adversary can forge a MAC is equal to guessing the key of the honest party $\frac{1}{|\mathbb{F}_{2^\kappa}|}$ or negligible in the security parameter. Therefore the probability of this event occurring is negligible and the distributions of $\mathcal{H}_5$ and $\mathcal{H}_4$ are statistically indistinguishable.

Hybrid $\mathcal{H}_6$. In this hybrid $\mathcal{S}_6$ implements the remainder of the Setup of $\mathcal{S}_{\mathsf{thresh}}$ and no longer receives the input $(\texttt{setup}, \mathsf{sid}, \mathcal{C}, x^i)$ of the honest parties as $\mathcal{F}_{\mathsf{thresh}}$. However, like $\mathcal{S}_5$, $\mathcal{S}_6$ does not use knowledge of the honest input x^i in any way, and so this is purely a syntactic change. The distribution of $\mathcal{H}_6$ is identical to that of $\mathcal{H}_5$.

Hybrid $\mathcal{H}_7$. In this hybrid, $\mathcal{Z}$ now communicates with $\mathcal{S}_{\mathsf{thresh}}$ instead of $\mathcal{S}_6$. This is a syntactic change and so $\mathcal{H}_7$ is distributed identically to $\mathcal{H}_6$.

By hybrid argument, the protocol π_{thresh} correctly implements the ideal functionality $\mathcal{F}_{\mathsf{thresh}}$, and the Real experiment is statistically indistinguishable from the Ideal experiment.

$$\texttt{Real}_{\pi_{\mathsf{thresh}}, \mathcal{A}, \mathcal{Z}}(\kappa, z)_{z \in \{0,1\}^*} \approx \texttt{Ideal}_{\mathcal{F}_{\mathsf{thresh}}, \mathcal{S}_{\mathsf{thresh}}, \mathcal{Z}}(\kappa, z)_{z \in \{0,1\}^*}$$

□

5 Cost Analysis

In this section we derive a model that can be used to estimate communication complexity. We derive asymptotic complexity figures using the scheme of [YWZ20] as the implementation of $\mathcal{F}_{\mathsf{aBit}}$ and $\mathcal{F}_{\mathsf{MPC}}$.

Claim 5.1 (Asymptotic Complexity of $\texttt{ThreshAbits}$)**.** *The per-party asymptotic cost to create ℓ threshold authenticated bits is $O(n(\ell + \kappa + s)\kappa + n^2\kappa)$.*

Claim 5.2 (Asymptotic Complexity of π_{thresh} Setup)**.** *Let $\mathcal{C}$ be a circuit with $\mathcal{I}$ input wires, G AND gates, and $\mathcal{O}$ output wires, then the per-party asymptotic cost of Setup in π_{thresh} is $O\left(n \cdot \kappa \cdot \left(\mathcal{I} + \frac{Gs}{\log G} + \mathcal{O} + \kappa\right) + n^2\kappa\right)$.*

Claim 5.3 (Asymptotic Complexity of π_{thresh} Eval)**.** *Let $\mathcal{C}$ be a circuit with $\mathcal{I}$ input wires, G AND gates, and $\mathcal{O}$ output wires, then the per-party asymptotic cost of Eval in π_{thresh} is $O\left(n \cdot \kappa \cdot \left(\mathcal{I} + \frac{Gs}{\log G} + \mathcal{O} + \kappa\right) + n^2\kappa\right)$.*

Claim 5.4 (Asymptotic Complexity of Folklore)**.** *Let $\mathcal{C}$ be a circuit with $\mathcal{I}$ input wires, G AND gates, and $\mathcal{O}$ output wires, then the per-party asymptotic cost of Setup in of the Folklore threshold protocol is $O\left(n \cdot \kappa \left(\mathcal{I} + (nt\mathcal{O}) + \frac{(G + \mathcal{O}\log s)s}{\log(G + \mathcal{O}\log s)} + \kappa\right) + n^2\kappa\right)$.*

These claims follow directly from by expanding taBitSampleCost and tMpcSetupCost defined below. We provide the equations that were used to derive these asymptotics in the following subsections.

5.1 Building Blocks

All functions are implicitly parameterized by the number of parties n, computational security parameter κ, and the statistical security parameter s. The $\mathcal{F}_{\mathsf{com}}$ and $\mathcal{F}_{\mathsf{rand}}$ functionalities are implemented simply as in 3.1.

$$\begin{aligned}\texttt{ComCost}(\ell) &\mapsto (n-1)(4\kappa+\ell)\\ \texttt{RandSetupCost} &= \texttt{ComCost}(\kappa)\end{aligned}$$

We instantiate our protocol with the OT extension protocol of [KOS15] using base OTs from Simplest OT [CO15] (as implemented in [WMK16]). We remark that the SoftSpoken OT extension protocol of [Roy22] with base OTs using the endemic OT protocol of [ZZZR23] can replace the usage of [KOS15] for improved performance and security. Analysis for each of these cases is provided, where ℓ is the number of instances, G the element size in the base OT, and m the size of the desired correlation.

$$\begin{aligned}\texttt{OTCost}_{CO}(\ell,G) &\mapsto 2\ell(G+\kappa)\\ \texttt{OTCost}_{ZZZR}(\ell,G) &\mapsto \frac{\kappa+(2\ell+1)G+(5\ell+2)G(\kappa/\log\kappa)}{2}\\ \texttt{COTeSetupCost}_{KOS} &= \texttt{OTCost}(\kappa,2\kappa)\\ \texttt{COTeSetupCost}_{Roy} &= \texttt{OTCost}(\kappa,2\kappa)+\kappa\\ \texttt{COTeCost}_{KOS}(\ell,m) &\mapsto \ell*(m+\kappa)+(2\kappa^2+4\kappa)\\ \texttt{COTeCost}_{Roy}(\ell,m) &\mapsto \ell*m/2+\frac{5}{8}(\kappa^2+\kappa)+\kappa*\ell/4\end{aligned}$$

The multiplication functionality is instantiated by the multiplication protocol of [DKLs19], and $\mathcal{F}_{\mathsf{is-zero}}$ by performing a random multiplication followed by opening the output shares.

$$\begin{aligned}\texttt{MultCost}(\zeta=\kappa+2s) &= (n-1)(\texttt{COTeCost}(2\zeta,2\kappa)+(6+\zeta)\kappa)\\ \texttt{IsZeroCost} &= \texttt{MultCost}+\texttt{ComCost}(\kappa)\end{aligned}$$

Cost of the Yang et al. [YWZ20] MPC protocol To complete the remaining base functionalities we describe the cost of the [YWZ20] protocol in terms of the base correlated OT extension protocol and the circuit to be executed. Internally the MPC protocol first runs a preprocessing phase to generate authenticated bits as well as authenticated Beaver triples depending on the number of input wires and AND gates in the circuit. Once the preprocessing is complete the protocol generates garbling labels for each of the gates in the circuit, and then takes inputs and evaluates the circuit. We use the notation

$C = (\mathcal{I}, G, \mathcal{O})$ to describe the number of inputs, AND gates, and outputs for a circuit C, and ℓ the number of instances to generate.

$$\begin{aligned}
\texttt{aBitCost}(\ell) &\mapsto 2(n-1)\texttt{COTeCost}(\ell + 2\kappa + s, \kappa) + 2(n^2+n)\kappa \\
\texttt{aAndCost}(\ell, B = 1 + \frac{s}{\log \ell + 1}) &\mapsto \texttt{aBitCost}(3\ell B) + 4(n-1)B\ell\kappa \\
\texttt{MPCCost}(C = (\mathcal{I}, G, \mathcal{O})) &\mapsto \texttt{aBitCost}(\mathcal{I} + G) + \texttt{aAndCost}(G) \\
&\quad + (2Gn + (2 + 4(n-2))G)\kappa + \mathcal{I}\kappa + (G + 3\kappa) + (n-1)\mathcal{O}\kappa
\end{aligned}$$

The costs of the protocol are overall dominated by the preprocessing to generate authenticated bits and triples, followed by the cost to send the garbled labels.

Threshold Authenticated Bits The `ThreshAbits` protocol has two components, initialization and sampling, and we describe the cost of each of these components. Initialization solely consists of running the correlated OT setup for the $\mathcal{F}_{\mathsf{aBit}}$ protocol. Sampling is the cost of sampling random threshold shares, authenticating the bit decomposition, and then evaluating the consistency check using the multiplication protocol.

$$\begin{aligned}
\texttt{taBitSetupCost} &= 2(n-1)\texttt{COTeSetupCost} \\
\texttt{taBitSampleCost}(\ell, B = \lceil \ell/\kappa \rceil) &\mapsto \texttt{aBitCost}(B\kappa) + 2(n-1)B\kappa + \texttt{IsZeroCost}
\end{aligned}$$

5.2 Our Threshold MPC Protocol Costs

Here, we combine the analysis from above to analytically describe the full costs associated with our main $\mathcal{F}_{\mathsf{thresh}}$ protocol.

As mentioned in the previous section for $\mathcal{F}_{\mathsf{MPC}}$, we use $C = (\mathcal{I}, G, \mathcal{O})$ to describe the size of the circuit being executed. The setup cost consists of the underlying OT extension setup, sampling threshold authenticated bits of an output mask, and evaluating the setup circuit. Evaluating a circuit with a threshold-shared input is simply the cost of the underlying MPC evaluation (with $t \leq n$ parties) as there is no other communication.

$$\begin{aligned}
\texttt{tMPCSetupCost}(C = (\mathcal{I}, G, \mathcal{O})) &\mapsto \texttt{taBitSetupCost} + \texttt{taBitSampleCost}(\mathcal{O}) \\
&\quad + \texttt{MPCCost}((\mathcal{I} + \mathcal{O}, G, \mathcal{O})) \\
\texttt{tMPCEvalCost}(C) &\mapsto \texttt{MPCCost}(C)
\end{aligned}$$

For a single execution there is small overhead running setup followed by evaluation over the plain non-threshold MPC due to per-run setup costs and threshold resharing. This is most notable with AES because of its small circuit size and increased input size after preprocessing. Regardless as soon as 2 calls to Eval are performed it is immediately advantageous, ignoring any complexity of ensuring consistency of inputs using just the plain $\mathcal{F}_{\mathsf{MPC}}$.

5.3 Comparison with folklore methods

Recall the main difference between our approach and the folklore is that the folklore approach requires integrating the use of the threshold secrets *inside* the MPC evaluation of the circuit.

There are several design choices to make, and we attempt to choose the best options to maintain a fair comparison. In particular, we compare to the scheme where each party $\mathcal{P}^i$ holds a t-of-n threshold share of global key $[\![\alpha]\!]^i$, each input $[\![\boldsymbol{x}]\!]^i$, and MACs on each input $[\![\boldsymbol{m} = \alpha\boldsymbol{x}]\!]^i$. While many MPC protocols use prime-order finite fields, here we specifically choose the field $\mathbb{F}_{2^\kappa}$ as field operations have (relatively) low boolean circuit complexity. As a result, reconstructing a secret-shared element and computing addition in $\mathbb{F}_{2^\kappa}$ solely consists of XOR gates and does not contribute to the AND complexity of the circuit.

The multiplication over $\mathbb{F}_{2^\kappa}$ has AND complexity in $O(\kappa \log \kappa)$ using the fast Fourier transform to compute a polynomial multiplication. In particular the multiplication circuit for $\mathbb{F}_{2^\kappa}$ with $\kappa = 128$ used in the Folklore circuit is of size 3888 AND gates. The circuit calculates a multiplication αx for each field element x that is output.

We also need to create output secret shares $[\![\boldsymbol{x}]\!]^i$ and $[\![\alpha\boldsymbol{x}]\!]^i$ which is done by having each party input a random degree $t-1$ polynomial p^i for each output share, or $2t$ field elements for each $x, \alpha x$ pair. The circuit then calculates $p = \sum p^i$ being just XOR gates, and sets $p(0) = x$ or $p(0) = \alpha x$ as desired. With the random polynomial p representing the secret shares of x, then with the coefficients of p calculating $[\![x]\!]^i = p(i)$ is simply a constant multiplication and only XOR gates.

Letting $\mathcal{C} = (\mathcal{I}, G, \mathcal{O})$ describe a boolean circuit with $\mathcal{I}$ input wires, G AND gates, and $\mathcal{O}$ output wires, then the compiled setup circuit $\mathcal{C}'$ for n parties with threshold t has $\mathcal{I}' = \mathcal{I} + n(2t\mathcal{O} + \kappa)$ input wires, $G' = G + \mathcal{O}\log\kappa$ AND gates, and $\mathcal{O}' = 2n\mathcal{O}$ output wires. Likewise the compiled Eval circuit $\mathcal{C}_2'$ has $\mathcal{I}_2' = n(2\mathcal{I} + \kappa), G_2' = G + \mathcal{I}\log\kappa, \mathcal{O}_2' = \mathcal{O}$.

$$\texttt{folkMPCSetupCost}(\mathcal{C} = (\mathcal{I}, G, \mathcal{O})) \mapsto \texttt{MPCCost}((\mathcal{I} + n(2t\mathcal{O} + \kappa), G + \mathcal{O}\log\kappa, 2n\mathcal{O}))$$
$$\texttt{folkMPCEvalCost}(\mathcal{C} = (\mathcal{I}, G, \mathcal{O})) \mapsto \texttt{MPCCost}((n(2\mathcal{I} + \kappa), G + \mathcal{I}\log\kappa, \mathcal{O}))$$

Comparing to our method we see that the folklore method has substantially larger circuit sizes with additional multiplicative terms of $\Omega(n)$ for inputs and outputs, and $\log\kappa$ for AND gates. Thus directly the folklore method results in more expensive overall computation.

6 Benchmarks

We created a proof-of-concept implementation of our protocol in roughly 10 thousand lines of Rust code, including unit tests and comments. [1] Our implementation builds on the EMP toolkit developed by Wang, Malezmoff and

[1] The code for the implementation is available at https://github.com/Rosefield/thresh_mpc

Katz [WMK16] and uses their implementation of the authenticated garbling protocols of Wang et al., and Yang et al. [WRK17, YWZ20] as well as the (updated) malicious correlated OT extension protocol of Keller, Orsini, and Scholl [KOS15].

For each Setup or Eval operation being executed, our benchmarking programs establish insecure connections among the parties and then measures the wall clock time to complete the operation. Thus, they measure the impact of latency, computation, and bandwidth constraints. We also count bytes sent, including session identifier and data serialization overheads (but not authentication overheads, TCP overheads, etc.). In all cases, we compiled our code against nightly Rust 1.75 and set $\kappa = 128$ and $s = 80$.

We benchmarked our implementation using a set of Google Cloud Platform (GCP) `c2d-highcpu-8` nodes, running Debian 12 with kernel 6.1.52. Each node of this type has an EPYC 7B13 CPU from the AMD Epyc Milan family, with four physical cores clocked at 2.45 GHz that can execute eight threads in total, and 16GB of RAM. Every party in an experiment was allocated one node in the `us-east1-c` region.

6.1 Sample Applications

Our evaluation involves benchmarking threshold implementations of three cryptographic functions: AES, HMAC, and KMAC, in both setup for the threshold key, and evaluation of the function. In each case the $\texttt{Setup}(r^1, \ldots, r^n)$ function takes in key randomness from each party and, sums the randomness to create the initial key, and then derives some preprocessing from said key. Likewise the $\texttt{Eval}_k(m)$ function is parameterized by the shared preprocessing and calculates the encryption/MAC of the message.

In our AES benchmark, the Setup stage computes the AES key schedule $K' \leftarrow \texttt{Setup}^{AES}(r^1, \ldots r^n) = \texttt{AES-Schedule}(K = \sum_i r^i)$, and the Eval stage computes $\texttt{Eval}^{AES}_{K'}(m) = \texttt{AES-Expanded}_{K'}(m)$ on the message using the expanded key schedule. [2].

Our HMAC-SHA256 benchmark computes $H(K + \mathit{opad} || H(K + \mathit{ipad} || m))$ where K is a key, m is the message, and *opad* and *ipad* are constants. The Setup stage computes the first two key-dependent blocks $(s_1, s_2) \leftarrow \texttt{Setup}^{HMAC}(r^1, \ldots, r^n) = (\texttt{SHA}(IV, \mathit{ipad} + \sum_i r^i), \texttt{SHA}(IV, \mathit{opad} + \sum_i r^i))$ The Eval stage computes the final tag $t \leftarrow \texttt{Eval}^{HMAC}_{(s_1,s_2)}(m) = \texttt{SHA}(s_2, \texttt{SHA}(s_1, m))$.

Finally, the KMAC128(K, M) benchmark computes the KMAC tag on one block as $f(f(IV + K) + M)$ where f is the Keccak permutation of size 1600 and IV is the 1600-bit initial state for KMAC. The Setup stage computes the intermediate state $s \leftarrow \texttt{Setup}^{KMAC}(r^1, \ldots, r^n) = f(IV + \sum_i r^i)$. The Eval function computes the final tag $t \leftarrow \texttt{Eval}^{KMAC}_s(m) = f(s + m)_{[0,256]}$ that is the truncation of the permutation applied to the state and message.

[2]We note that in this specific case, the strategy of evaluating the key schedule during the setup leads to increased overhead (for the folklore method). Here it would be better to re-evaluate the key schedule instead of authenticating the larger number of bits, but we evaluate it nonetheless to compare the same design.

	Original Circuit			Folklore circuit	
	Inputs	$\wedge$-Gates	Outputs	$\wedge$-Gates	Blowup
HMAC-SHA256 Setup	256	45.15k	512	68.47k	1.51x
HMAC-SHA256 Eval	512	45.15k	256	61.47k	1.36x
AES128 Setup	128	1360	1408	44.05k	32.4x
AES128 Eval	1408	5440	128	49.74k	9.14x
KMAC128 Setup	256	38.40k	1600	96.72k	2.52x
KMAC128 Eval	1600	38.40k	256	90.86k	2.36x

Figure 1: Circuit complexity for different symmetric primitives. Each primitive is split into solely key-dependent and message-dependent components. The left columns indicate the circuit sizes for the standard version of the circuits. The right columns indicate the circuit size after applying the folklore threshold transformations; we apply the optimization of using the field $\mathbb{F}_{2^\kappa}$ to make the cost of Shamir reconstruction essentially XOR gates; nonetheless, the MAC computation and secret sharing steps requires some extra AND gates.

After implementing these functions, we compute their original size and benchmark the MPC performance for evaluating them using the state-of-the-art EMP toolkit. We compare the performance of the threshold schemes to a baseline evaluation of each function $F_k(m) = \texttt{Eval}_{\texttt{Setup}(\cdots)}(m)$ without any preprocessing or threshold resharing.

In addition, we also evaluate the blowup induced by applying the Folklore threshold method to these functions. Importantly, the folklore setup function must compute MACs on each batched-output value and additionally produce secret shares of the output $[\![x]\!]$ and the MAC $[\![\alpha x]\!]$. Calculating the MACs requires additional AND gates proportional to $\mathcal{O}\log\kappa$. Notably, creating output shares requires producing evaluations of a polynomial at n points, however, one can evaluate the polynomial at n fixed points using multiplications with field constants, which can be compiled into XOR gates and no extra AND gates. In contrast, our method only requires an additional $\mathcal{O}$ input wires and $\mathcal{O}$ XOR gates to mask the output of the circuit.

Figure 1 provides the original (non-threshold) circuit sizes for each these applications. As the table shows, the folklore method requires several thousand extra gates in each application, representing between 1.36–32x overhead.

Inputs In addition to the increase in the number of AND and XOR gates, the folklore method also incurs overhead by increasing the number of inputs in both the Setup and Eval circuit. In particular, each party inputs its own share of the keystate and MAC of the keystate during the evaluation, and thus the protocol overhead grows with the number n of participants in the protocol. The multiplication circuit for $\mathbb{F}_{2^\kappa}$ used in the Folklore circuit is of size 3888 AND gates. Figure 2 shows this factor of overhead can quickly approach or exceed

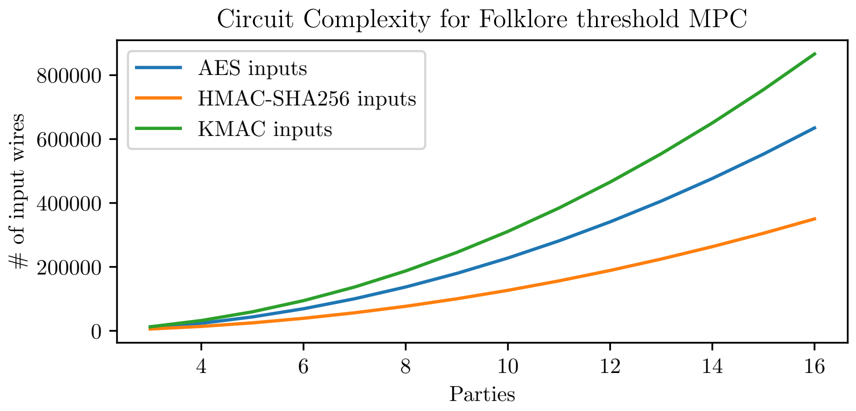

Figure 2: Number of input wires required by the folklore MPC method Setup as number of parties increases, with $t = n - 2$. At roughly $n = 4$, the increase in the number of input wires already overwhelms the original size of the circuit for all three of our applications.

the size of the modified circuit.

6.2 Timing and Communication Results

For each of our sample applications and for $n \in [3, 16]$ with $t = n - 1$ we collected the execution statistics for both our π_{thresh} and the Folklore solution, as well as a Baseline n-party evaluation of the underlying $\mathcal{F}_{\mathsf{MPC}}$ protocol for the full circuit. The collected statistics consist of the execution time in milliseconds and the number of bytes in megabytes, and we show the average of 30 runs for each data point. Timing and communication include the total cost of each evaluation including any data-independent "preprocessing" (such as base OTs) to generate the output of each circuit evaluation. During Setup each of the protocols executed the appropriate preprocessing circuit described in Fig. 1. Then during Eval our protocol and the Folklore protocol evaluated the circuit with preprocessed state. We compare the Setup and Eval costs as well as a combined one Setup+Eval against the cost of simply evaluating the application without any threshold sharing or preprocessing. Comparing Eval to the Baseline execution we show the advantage of preprocessing, and comparing Setup+Eval against Baseline shows the overhead of generating and using the threshold shares. For each protocol the cost of Eval depends on the number of parties actually evaluating, and so we report the cost of n parties running the Eval in a hypothetical n-of-X threshold sharing.

AES As per Fig. 3, our threshold protocol's time and communication performance closely matches the baseline [WRK17, YWZ20] implementation for evaluating the AES circuit, which supports our title claim that our approach to

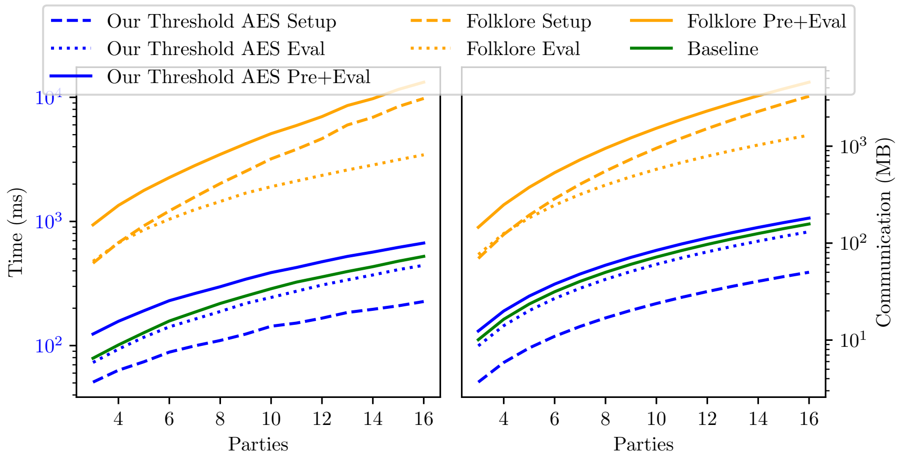

Figure 3: Computation time and communication for threshold AES evaluation for $\kappa = 128$. Our Setup and Evaluation times are just slightly more expensive than the Baseline MPC for computing AES (due to the extra work outside of MPC to Setup authenticated shares). In contrast, the Folklore methods are almost 1 order of magnitude slower. By re-using the preprocessing from Setup, our Thresh Eval circuit is only 5440 gates compared to the 6800 gates for the full AES circuit in the Baseline Eval.

threshold boolean functions is achieved at barely no overhead. In contrast, the folklore method requires roughly 10x overhead in both time and computation. This is due to the large increase in the number of input wires and the MAC computation required by folklore.

At a high level for online evaluation, with a threshold of 3 parties can evaluate $\approx$ 14/s AES blocks with $\approx$9MB communication each; or with a threshold of 16, $\approx$ 2.3/s evaluations with $\approx$135MB communication each. This gives a savings of $\approx$ 15% comp./comm. over the Baseline circuit for 3 parties, and $\approx$ 10% comp. and 17% comm. with 16 parties.

HMAC-SHA256 In the case of HMAC, the pre-processing step is beneficial for both the Folklore and our protocol. Specifically, the Setup calculates two SHA evaluations that include HMAC constants, while the Eval circuit uses these pre-computed values as the starting state, then computes 1 SHA block to incorporate the message, and then another SHA block to compute the output.

As a result, both the Folklore and our protocol outperform the Baseline circuit during Eval. However, as shown in Fig. 4, our implementation runs faster in Eval than both Folklore and Baseline, and in Setup the Folklore method considering runtime and communication.

Again for the online evaluation with a threshold of 3 parties can evaluate $\approx$ 2.3/s HMAC-SHA256 MACs with $\approx$68MB communication each; or with a

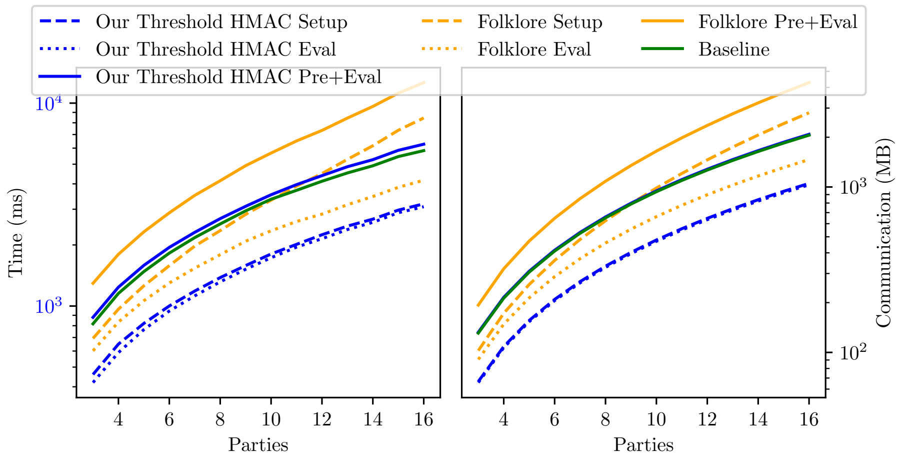

Figure 4: Computation time and communication for threshold HMAC-SHA256 evaluation. In this example we continue to see that our threshold scheme closely matches the baseline. Due to the increased circuit size and decreased number of outputs the Folklore method is relatively more competitive, but still is at least 50% slower. The preprocessing advantage is more pronounced for HMAC compared to AES as half of the entire circuit (45k gates) is removed during evaluation. Because of this both our protocol and the Folklore protocol (61k gates) outperform the Baseline evaluation (90k gates), with performance relative to the size of the circuit.

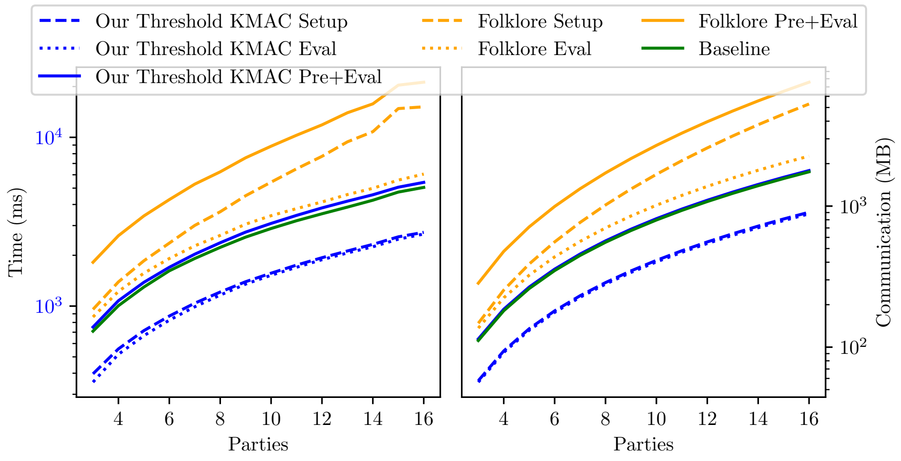

Figure 5: Computation time and communication for threshold KMAC evaluation. In this example we see that the Folklore method performs poorly in both Setup and Eval because of the increased state size (1600bits) causing the MAC creation and validation to dominate the cost of the KMAC circuit itself. As our threshold resharing is comparatively light the cost of evaluating the KMAC Setup circuit dominates and we closely match the cost of the Baseline during Setup and surpass it during Eval.

threshold of 16, ≈ 0.32/s evaluations with ≈1GB communication each. This gives a savings of ≈ 50% comp./comm. over the Baseline circuit for 3 parties, and ≈ 48% comp. and 50% comm. with 16 parties.

KMAC Finally, in Fig. 5 we evaluate the performance of our implementation on the KMAC circuit because it further demonstrates the applicability of our protocol. Our results here are comparable to what was achieved with HMAC, closely matching the cost of the Baseline circuit during Setup and surpassing the Baseline during Eval and the Folklore method in all cases.

In the last example the online evaluation with a threshold of 3 parties can evaluate ≈ 3.7/s KMAC128 MACs with ≈59MB communication each; or with a threshold of 16, ≈ 0.37/s evaluations with ≈924MB communication each. This gives a savings of ≈ 50% comp./comm. over the Baseline circuit for 3 parties, and ≈ 47% comp. and 50% comm. with 16 parties.

Acknowledgements

The authors of this work were supported by the NSF under grants 1646671, 1816028, and 2055568.

References

[ACE$^+$21] Mark Abspoel, Ronald Cramer, Daniel Escudero, Ivan Damgård, and Chaoping Xing. Improved single-round secure multiplication using regenerating codes. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 222–244, Cham, 2021. Springer International Publishing.

[AFSH$^+$20] Jackson Abascal, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Is the classical gmw paradigm practical? the case of non-interactive actively secure 2pc. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1591–1605, New York, NY, USA, 2020. Association for Computing Machinery.

[ANO$^+$22] Damiano Abram, Ariel Nof, Claudio Orlandi, Peter Scholl, and Omer Shlomovits. Low-bandwidth threshold ecdsa via pseudorandom correlation generators. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2554–2572, 2022.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BECO$^+$21] Aner Ben-Efraim, Kelong Cong, Eran Omri, Emmanuela Orsini, Nigel P. Smart, and Eduardo Soria-Vazquez. Large scale, actively secure computation from lpn and free-xor garbled circuits. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 33–63, Cham, 2021. Springer International Publishing.

[BGGK17] Dan Boneh, Rosario Gennaro, Steven Goldfeder, and Sam Kim. A lattice-based universal thresholdizer for cryptographic systems. 2017. https://eprint.iacr.org/2017/251.

[BLN$^+$21] Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High-performance multi-party computation for binary circuits based on oblivious transfer. *Journal of Cryptology*, 34(3):34, 2021.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 503–513, New York, NY, USA, 1990. Association for Computing Machinery.

[Can20] Ran Canetti. Universally composable security. *J. ACM*, 67(5), sep 2020.

[CCK23] Jung Hee Cheon, Wonhee Cho, and Jiseung Kim. Improved universal thresholdizer from threshold fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/545, 2023. https://eprint.iacr.org/2023/545.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 3–22, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[CCL$^+$20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold ecdsa. Cryptology ePrint Archive, Paper 2020/084, 2020. https://eprint.iacr.org/2020/084.

[CDK$^+$22] Megan Chen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, Abhi Shelat, and Ran Cohen. Multiparty generation of an rsa modulus. *Journal of Cryptology*, 35(2):12, 2022.

[CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based mpc protocol for boolean circuits with good amortized complexity. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography*, pages 652–682, Cham, 2020. Springer International Publishing.

[CGG$^+$20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1769–1787, New York, NY, USA, 2020. Association for Computing Machinery.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Proceedings of the 4th International Conference on Progress in Cryptology – LATINCRYPT 2015 - Volume 9230*, page 40–58, Berlin, Heidelberg, 2015. Springer-Verlag.

[CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 35–67, Cham, 2023. Springer Nature Switzerland.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy

Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 57–87, Cham, 2022. Springer Nature Switzerland.

[DK01] Ivan Damgård and Maciej Koprowski. Practical threshold rsa signatures without a trusted dealer. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 152–165, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[DKL$^+$12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. In Ivan Visconti and Roberto De Prisco, editors, *Security and Cryptography for Networks*, pages 241–263, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ecdsa from ecdsa assumptions; the multiparty case. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1051–1066, 2019.

[DKSS13] Zeev Dvir, Swastik Kopparty, Shubhangi Saraf, and Madhu Sudan. Extensions to the method of multiplicities, with applications to kakeya sets and mergers. *SIAM Journal on Computing*, 42(6):2305–2328, 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[EXY22] Daniel Escudero, Chaoping Xing, and Chen Yuan. More efficient dishonest majority secure computation over z2k via galois rings. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part I*, page 383–412, Berlin, Heidelberg, 2022. Springer-Verlag.

[FKOS15] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to mpc with preprocessing using ot. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 711–735, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1179–1194, New York, NY, USA, 2018. Association for Computing Machinery.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[HIV17] Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 3–39, Cham, 2017. Springer International Publishing.

[HKO23] David Heath, Vladimir Kolesnikov, and Rafail M. Ostrovsky. Tristate circuits - a circuit model that captures ram. In *Annual International Cryptology Conference*, 2023.

[HSSV17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round mpc combining bmr and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 598–628, Cham, 2017. Springer International Publishing.

[KMOS21] Yashvanth Kondi, Benardo Magari, Claudio Orlandi, and Omer Shlomovitz. Refresh when you wake up: Proactive threshold wallets with offline device. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 608–625, 2021.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure ot extension with optimal overhead. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, pages 724–741, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 830–842, New York, NY, USA, 2016. Association for Computing Machinery.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 365–391, Cham, 2018. Springer International Publishing.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1837–1854, New York, NY, USA, 2018. Association for Computing Machinery.

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 495–512, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[LPSY19] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant-round multi-party computation combining bmr and spdz. *Journal of Cryptology*, 32(3):1026–1069, 2019.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 681–700, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[oST23] National Institute of Standards and Technology. Fips 186-5:digital signature standard (dss). 2023.

[Roy22] Lawrence Roy. Softspokenot: Quieter OT extension from small-field silent VOLE in the minicrypt model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 657–687. Springer, 2022.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.

[RW19] Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology – INDOCRYPT 2019*, pages 227–249, Cham, 2019. Springer International Publishing.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 207–220, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

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

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 39–56, New York, NY, USA, 2017. Association for Computing Machinery.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient mpc from improved triple generation and authenticated garbling. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1627–1646, New York, NY, USA, 2020. Association for Computing Machinery.

[ZCSH18] Ruiyu Zhu, Darion Cassel, Amr Sabry, and Yan Huang. Nanopi: Extreme-scale actively-secure multi-party computation. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 862–879, New York, NY, USA, 2018. Association for Computing Machinery.

[ZZZR23] Zhelei Zhou, Bingsheng Zhang, Hong-Sheng Zhou, and Kui Ren. Endemic oblivious transfer via random oracles, revisited. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 303–329, Cham, 2023. Springer Nature Switzerland.

A Example of Threshold Authenticated Bit Conversion

Consider the following simplified example for the definition of the family of functions $f_*^{*;*}$.

Example A.1. *Let* $\mathbb{F}_{2^4} = \mathbb{F}_2[X]/\langle X^4+X+1\rangle$*, and say that* $\lambda_1^S(0) = X^3+X^2+1$ *with a set of parties* S *with* $1 \in S$*. Then for some element* a*, with* a_i *the* i^{th} *bit* *of a, we have*

$$\begin{aligned}
\lambda_1^S(0) \cdot a &= a + X^2 a + X^3 a \\
&= a_4 X^3 + a_3 X^2 + a_2 X + a_1 \\
&+ a_4(X^5 = X(X+1)) + a_3(X^4 = (X+1)) + a_2(X^3) + a_1(X^2) \\
&+ a_4(X^6 = X^2(X+1)) + a_3(X^5 = X(X+1)) + a_2(X^4 = (X+1)) + a_1(X^3) \\
&= (2a_4 + a_2 + a_1)X^3 + (2a_3 + 2a_4 + a_1)X^2 + (2a_2 + a_4 + 2a_3)X + (a_1 + a_3 + a_2) \\
&= (a_2 + a_1)X^3 + (a_1)X^2 + (a_4)X + (a_1 + a_3 + a_2)
\end{aligned}$$

If we only care about the 1st bit then $\texttt{Bits}(\lambda_1^S(0) \cdot a)_1 = a_1 + a_3 + a_2$ and $f_1^{1,S} : \boldsymbol{x} \mapsto x_1 + x_3 + x_2$. Observe that

$$\begin{aligned}
f_1^{1,S}(M_j^1[\llbracket x \rrbracket^1]) &= (M_j^1[b_1^1] + M_j^1[b_3^1] + M_j^1[b_2^1]) \\
&= (K^j[b_1^1] + b_1^1 \Delta^j) + (K^j[b_3^1] + b_3^1 \Delta^j) + (K^j[b_2^1] + b_2^1 \Delta^j) \\
&= f_1^{1,S}(K^j[\llbracket x \rrbracket^1]) + (b_1^1 + b_3^1 + b_2^1)\Delta^j
\end{aligned}$$

is a valid MAC for $\texttt{Bits}(\lambda_1^S(0) \cdot \llbracket x \rrbracket^1)_1$.