Concretely Efficient Large-Scale MPC with Active Security (or, TinyKeys for TinyOT)

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

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

Abstract. In this work we develop a new theory for concretely efficient, large-scale MPC with active security. Current practical techniques are mostly in the strong setting of all-but-one corruptions, which leads to protocols that scale badly with the number of parties. To work around this issue, we consider a large-scale scenario where a small minority out of many parties is honest and design scalable, more efficient MPC protocols for this setting. Our results are achieved by introducing new techniques for information-theoretic MACs with short keys and extending the work of Hazay et al. (CRYPTO 2018), which developed new passively secure MPC protocols in the same context. We further demonstrate the usefulness of this theory in practice by analyzing the concrete communication overhead of our protocols, which improve upon the most efficient previous works.

1 Introduction

Secure multi-party computation (MPC) protocols allow a group of n parties to compute some function f on the parties' private inputs, while preserving a number of security properties such as *privacy* and *correctness*. The former property implies data confidentiality, namely, nothing leaks from the protocol execution but the computed output. The

⋆ Supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS), and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.
⋆⋆ Supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT.
⋆⋆⋆ Supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583 (SODA), and the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC).
† Supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 643161, and by ERC Advanced Grant ERC-2015-AdG-IMPaCT.

latter requirement implies that the protocol enforces the integrity of the computations made by the parties, namely, honest parties are not led to accept a wrong output. Security is proven either in the presence of a passive adversary that follows the protocol specification but tries to learn more than allowed from its view of the protocol, or an active adversary that can arbitrarily deviate from the protocol specification in order to compromise the security of the other parties in the protocol.

The past decade has seen huge progress in making MPC protocols communication efficient and practical; see [KS08, DPSZ12, DKL+13, ZRE15, LPSY15, WMK17, HSS17] for just a few examples. In the two-party setting, actively secure protocols [WRK17a] by now reach within a constant overhead factor over the notable semi-honest construction by Yao [Yao86]. On the practical side, a Boolean circuit with around 30,000 gates (6,400 AND gates and the rest XOR) can be securely evaluated with active security in under 20ms [WRK17a]. Moreover, current technology already supports protocols that securely evaluate circuits with more than a billion gates [KSS12]. On the other hand, secure *multi-party* computation with a larger number of parties and a dishonest majority is far more difficult due to scalability challenges regarding the number of parties. Here, the most efficient practical protocol with active security has a multiplicative factor of $O(\lambda/\log|C|)$ due to cut-and-choose [WRK17b] (where λ is a statistical security parameter and $|C|$ is the size of the computed circuit). On the practical side, the same Boolean circuit of 30,000 gates can be securely evaluated at best in 500ms for 14 parties [WRK17b] in a local network where the latency is neglected, or in more than 20s in a wide network. The problem is that current MPC protocols do not scale well with the number of parties, where the main bottleneck is a relatively high communication complexity, while the number of applications requiring large scale communication networks are constantly increasing, involving sometimes hundreds of parties.

An interesting example is safely measuring the Tor network [DMS04] which is among the most popular tools for digital privacy, consisting of more than 6000 relays that can opt-in for providing statistics about the use of the network. Nowadays and due to privacy risks, the statistics collected over Tor are generally poor: There is a reduced list of computed functions and only a minority of the relays provide data, which has to be obfuscated before publishing [DMS04]. Hence, the statistics provide an incomplete picture which is affected by a noise that scales with the number of relays.

In the context of securely computing the interdomain routing within the Border Gateway Protocol (BGP) which is performed at a large scale of thousands of nodes, a recent solution in the dishonest majority setting [ADS+17] centralizes BGP so that two parties run this computation for all Autonomous Systems. Large scale protocols would allow scaling to a large number of systems computing the interdomain routing themselves using MPC, hence further reducing the trust requirements.

Another important application that involves a massive number of parties is an auction with private bids, where the winning bid is either the first or the second price. Auctions have been widely studied by different communities improving different aspects and are central in the area of web electronic commerce. When considering privacy and correctness, multi-party computation offers a set of tools that allow to run the auction while preserving the privacy of the bidders (aka. passive security). MPC can

also enforce independent of inputs between the corrupted and honest parties as well as correctness, in the sense that parties are not allowed to change their vote once they learn they lost. This type of security requires more complicated tools and is knows as active security. Designing secure solutions for auctions played an important role in the literature of MPC. In fact, the first MPC real-world implementation was for the sugar beet auction [BCD+09] with three parties and honest majority, where the actual number of parties was 1129. In a very recent work by Keller et al. [KPR18], the authors designed a new generic protocol based on semi-homomorphic encryption and lattice-based zero-knowledge proofs of knowledge, and implemented the second-price auction with 100 parties over a field of size 2^{40}. The running time of their offline phase for the SPDZ protocol is 98s. The authors did not provide an analysis of their communication complexity.

Motivated by the fact that current techniques are insufficient to produce highly practical protocols for such scenarios, we investigate the design of protocols that can more efficiently handle large numbers of parties with *strong security* levels. In particular, we study the setting of active security with only a minority (around 10–30%) of honest participants. By relaxing the well-studied, very strong setting of all-but-one corruptions (or *full-threshold*), we hope to greatly improve performance. Our starting point is the recent work by Hazay et al. [HOSS18] which studied this corruption setting with passive security and presented a new technique based on "short keys" to improve the communication complexity and the running times of full-threshold MPC protocols. In this paper we extend their results to the active setting.

Technical background for [HOSS18]. Towards achieving their goal, Hazay et al. observed that instead of basing security on secret keys held by each party individually, they can base security on the *concatenation of all honest parties' keys*. Namely, a secure multi-party protocol with h honest parties can be built by distributing secret key material so that each party only holds a *small part of the key*. Formalizing this intuition is made possible by reducing the security of their protocols to the *Decisional Regular Syndrome Decoding (DRSD)* problem, which, given a random binary matrix $\mathbf{H}$, is to distinguish between the syndrome obtained by multiplying $\mathbf{H}$ with an error vector $\boldsymbol{e} = (\boldsymbol{e}_1 \| \cdots \| \boldsymbol{e}_h)$ where each $\boldsymbol{e}_i \in \{0,1\}^{2^\ell}$ has Hamming weight one, and the uniform distribution. This can equivalently be described as distinguishing $\bigoplus_{i=1}^{h} \mathsf{H}(i, k_i)$ from the uniform distribution, where H is a random function and each k_i is a random ℓ-bit key. A specified in [HOSS18], when h is large enough, the problem is *unconditionally hard* even for $\ell = 1$, which means for certain parameter choices 1-bit keys can be used *without introducing any additional assumptions*.

Our contribution. In this work we develop a new theory for concretely efficient, large-scale MPC in the presence of an active adversary. More concretely, we extend the short keys technique from [HOSS18] to the active setting. Adapting these ideas to the active setting is quite challenging and requires modifying information-theoretic MACs used in previous MPC protocols [BDOZ11, DPSZ12] to be usable with short MAC keys. As our first, main contribution, we present several new methods for constructing efficient, distributed, information-theoretic MACs with short keys, for the setting of a small, honest minority out of a large set of parties. Our schemes allow for much lower costs when

creating MACs in a distributed manner compared with previous works, due to the use of short MAC keys. For our second contribution, we show how to use these efficient MAC schemes to construct actively secure MPC for binary circuits, based on the 'TinyOT' family of protocols [NNOB12, BLN+15, FKOS15, HSS17, WRK17b]. All previous protocols in that line of work supported $n-1$ out of n corruptions, so our protocol extends this to be more efficient for the setting of large-scale MPC with a few honest parties.

Concrete efficiency improvements. The efficiency of our protocols depends on the total number of parties, n, and the number of honest parties, h, so there is a large range of parameters to explore when comparing with other works. We discuss this in more detail in Section 8. Our protocol starts to concretely improve upon previous protocols when we reach $n = 30$ parties and $t = 18$ corruptions: here, our triple generation method requires less than *half the communication cost* of the fastest MPC protocol which is also based on TinyOT [WRK17b] (dubbed WRK) tolerating up to $n-1$ corruptions. For a fairer comparison, we also consider modifying WRK to run in a committee of size $t+1$, to give a protocol with the same corruption threshold as ours. In this setting, we see a small improvement of around 10% over WRK, but at larger scales the impact of our protocol becomes much greater. For example, with $n = 200$ parties and $t = 160$ corruptions we have up to an 8 times improvement over WRK with full-threshold, and a 5 times improvement when WRK is modified to the threshold-t setting.

Technical Overview

In our protocols we assume that two committees, $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$, have been selected out of all the n parties providing inputs in the MPC protocol, such that $\mathcal{P}_{(h)}$ contains at least h honest parties and $\mathcal{P}_{(1)}$ contains at least 1 honest party. These can be chosen deterministically, for instance, if there are h honest parties in total we let $\mathcal{P}_{(h)} = \{P_1, \ldots, P_n\}$ and $\mathcal{P}_{(1)} = \{P_1, \ldots, P_{n-h+1}\}$. We can also choose committees at random using coin-tossing, if we start with a very large group of parties from which $h' > h$ are honest. Since we have $|\mathcal{P}_{(h)}| > |\mathcal{P}_{(1)}|$, to avoid unnecessary interaction we take care to ensure that committee $\mathcal{P}_{(h)}$ is only used when needed, and when possible we will do operations in committee $\mathcal{P}_{(1)}$ only.

Section 3. We first show a method for authenticated secret-sharing based on information-theoretic MACs with short keys, where given a message x, a MAC $\boldsymbol{m}$ and a key $\boldsymbol{k}$, verification consists of simply checking that s linear equations hold. Our construction guarantees that forging a MAC to all parties can only be done with probability $2^{-\lambda}$, *even when the key length ℓ is much smaller than λ*, by relying on the fact that at least h parties are honest. We note that the reason for taking this approach is *not* to obtain a more efficient MAC scheme, but to design a scheme allowing more efficient *creation* of the MACs. Setting up the MACs typically requires oblivious transfer, with a communication cost proportional to the key length, so a smaller ℓ gives us direct efficiency improvements to the preprocessing phase, which is by far the dominant cost in applications. Our basic MAC scheme requires all parties in both committees to take part, but to improve this we also present several optimizations, which can greatly reduce the

storage overhead by "compressing" the MACs into a single, SPDZ-like sharing in *only* committee $\mathcal{P}_{(1)}$.

Section 4–5. We next show how to efficiently create authenticated shares for our MAC scheme with short keys. As a building block, we need a protocol for random correlated oblivious transfer (or random Δ-OT) on short strings. We consider a variant of the OT extension protocol of Keller et al. [KOS15], modified to produce correlated OTs (as done in [NST17]) and with short strings. Our authentication protocol for creating distributed MACs improves upon the previous best-known approach for creating MACs (optimized to use h honest parties) by a factor of $h(n-h)/n$ times in terms of overall communication complexity. This gives performance improvements *for all* $h > 1$, with a maximum $n/4$-fold gain as h approaches $n/2$.

Section 7. Finally, we introduce our triple generation protocol, in two phases. Similarly to [WRK17b], we first show how to compute the cross terms in multiplication triples by computing so-called 'half-authenticated' triples. This protocol does not authenticate all terms and the result may yield an incorrect triple. Next, we run a standard cut-and-choose technique for verifying correctness and removing potential leakage. Our method for checking correctness does not follow the improved protocol from [WRK17b] due to a limitation introduced by our use of the DRSD assumption. The security of our protocol relies on a variant of the DRSD assumption that allows one bit of leakage, and for this reason the number of triples r generated by these protocols depends on the security of RSD. So, while we can produce an essentially unlimited number of random correlated OTs and random authenticated bits, if we were to produce 'half-authenticated' triples in a naive way, we would be bounded on the total number of triples and hence the size of the circuits we can evaluate. To fix this issue we show how to switch the MAC representation from using one key Δ to a representation under another independent key $\tilde{\Delta}$. This switch is performed every r triples.

Extension to Constant Rounds. Since Hazay et al. [HOSS18] also described a constant round protocol based on garbled circuits with passive security, it is natural to wonder if our approach with active security also extends to this setting. Unfortunately, it is not straightforward to extend our approach to multi-party garbled circuits with short keys and active security, since the adversary can flip a garbled circuit key with non-negligible probability, breaking correctness. Nevertheless, we can build an alternative, efficient solution based on the transformation from [HSS17], which shows how to turn any non-constant round, actively secure protocol for Boolean circuits into a constant round [BMR90]-based protocol. When applying [HSS17] to our protocol, we obtain a multi-party garbling protocol with full-length keys, but we still improve upon the naive (full-threshold) setting, since the preprocessing phase is more efficient due to our use of TinyOT with short keys. More details will be given in the full version.

2 Preliminaries

We denote the computational and statistical security parameter by κ and λ, respectively. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large κ it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. The function μ is *noticeable* (or

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

Security and Communication Models. We use the universal composability (UC) framework [Can01] to analyse the security of our protocols. We assume all parties are connected via secure, authenticated point-to-point channels, as well as a broadcast channel which is implemented using a standard 2-round echo-broadcast. The adversary model we consider is a static, active adversary who corrupts up to t out of n parties at the beginning of the protocol. We denote by A the set of corrupt parties, and $\bar{A}$ the set of honest parties.

Regular Syndrome Decoding Problem. We recall that the *regular syndrome decoding* (RSD) problem is to recover a secret error vector $\boldsymbol{e} = (\boldsymbol{e}_1 \| \cdots \| \boldsymbol{e}_h)$, where each $\boldsymbol{e}_i \in \{0,1\}^{m/h}$ has Hamming weight one, given only $(\mathbf{H}, \mathbf{H}\boldsymbol{e})$, for a randomly chosen binary $r \times m$ matrix $\mathbf{H}$. In [HOSS18] it was shown that the search and decisional versions of this problem are equivalent and even statistically secure when h is big enough compared to r. In this work we use an interactive variant of the problem, where the adversary is allowed to try to guess a few bits of information on the secret $\boldsymbol{e}$ before seeing the challenge; if the guess is incorrect, the game aborts. We conjecture that this 'leaky' version of the problem, defined below, is no easier than the standard problem. Note that on average the leakage only allows the adversary to learn 1 bit of information on $\boldsymbol{e}$, since if the game does not abort he only learns that $\bigwedge P_i(\boldsymbol{e}) = 1$.

The 'leaky' part of the assumption is introduced as a result of an efficient instantiation of random correlated OTs on short strings (Section 4). Once the adversary has tried to guess these short strings, which act as short MAC keys in the authentication protocol (Section 5), a DRSD challenge is presented to him during the protocol computing the cross terms of multiplication triples (Section 7.1). As in [HOSS18], the appearance of the DRSD instance is due to the fact of 'hashing' the short MAC keys of at least h honest parties during said multiplications.

Definition 2.1 (Decisional Regular Syndrome Decoding with Leakage) *Let $r, h, \ell \in \mathbb{N}$ and $m = h \cdot 2^\ell$. Consider the game $\mathcal{L}$-$\mathsf{DRSD}^b_{r,h,\ell}$ for $b \in \{0,1\}$, defined between a challenger and an adversary:*

1. *Sample $\mathbf{H} \leftarrow \mathbb{F}_2^{r \times m}$ and a random, weight-h vector $\boldsymbol{e} \in \mathbb{F}_2^m$.*
2. *Send $\mathbf{H}$ to the adversary and wait for the adversary to adaptively query up to h efficiently computable[5] predicates $P_i : \mathbb{F}_2^m \to \{0,1\}$. For each P_i queried, if $P_i(\boldsymbol{e}) = 0$ then abort, otherwise wait for the next query.*

[5] By efficiently computable, we mean that the adversary sends a description of a polynomially-sized circuit that computes P.

3. *If* $b = 0$*, sample* $\boldsymbol{u} \leftarrow \mathbb{F}_2^r$ *and send* $(\mathbf{H}, \boldsymbol{u})$ *to the adversary. Otherwise if* $b = 1$*, send* $(\mathbf{H}, \mathbf{H}\boldsymbol{e})$.

The DRSD problem with leakage *with parameters* (r, h, ℓ) *is to distinguish between* $\mathcal{L}\text{-}\mathsf{DRSD}^0_{r,h,\ell}$ *and* $\mathcal{L}\text{-}\mathsf{DRSD}^1_{r,h,\ell}$ *with noticeable advantage.*

2.1 Resharing

At several points in our protocols, we have a value $x = \sum_{i \in X} x^i$ that is secret-shared between a subset of parties $\{P_i\}_{i \in X}$, and wish to re-distribute this to a fresh sharing amongst a different set of parties, say $\{P_j\}_{j \in Y}$. The naive method to do this is for every party P_i to generate a random sharing of x^i, and send one share to each P_j. This costs $|X| \cdot |Y| \cdot m$ bits of communication, where m is the bit length of x. When m is large, we can optimize this using a pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^m$, as follows:

1. For $i \in X$, party P_i does as follows:
   (a) Pick an index $j' \in Y$[6]
   (b) Sample random keys $k^{i,j} \leftarrow \{0,1\}^\kappa$, for $j \in Y \setminus j'$
   (c) Send $k^{i,j}$ to party P_j, and send $x^{i,j'} = \sum_j G(k^{i,j}) + x^i$ to party $P_{j'}$
2. For $j \in Y$, party P_j does as follows:
   (a) Receive $k^{i,j}$ from each P_i who sends P_j a key, and a share $x^{i,j}$ from each P_i who sends P_j a share. For the keys, compute the expanded share $x^{i,j} = G(k^{i,j})$
   (b) Output $x^j = \sum_{i \in X} x^{i,j}$.

Now each P_i only needs to send a single share of size m bits, since the rest are compressed down to κ bits using the PRG. This gives an overall communication complexity of $O(|X| \cdot |Y| \cdot \kappa + |X| \cdot m)$ bits.

3 Information-Theoretic MACs with Short Keys

We now describe our method for authenticated secret-sharing based on information-theoretic MACs with short keys. Our starting point is the standard information-theoretic MAC scheme on a secret $x \in \{0,1\}$ given by $\boldsymbol{m} = \boldsymbol{k} + x \cdot \Delta$, for a uniformly random key $(\boldsymbol{k}, \Delta)$, where $\boldsymbol{k} \in \{0,1\}^\ell$ is only used once per message x, whilst $\Delta \in \{0,1\}^\ell$ is fixed. Given the message x, the MAC $\boldsymbol{m}$ and the key $\boldsymbol{k}$, verification consists of simply checking the linear equation holds. It is easy to see that, given x and $\boldsymbol{m}$, forging a valid MAC for a message $x' \neq x$ is equivalent to guessing Δ. In a nutshell, we adapt this basic scheme for the multi-party, secret-shared setting, with the guarantee that forging a MAC to all parties can only be done with probability $2^{-\lambda}$, *even when the key length* ℓ *is much smaller than* λ, by relying on the fact that at least h parties are honest.

Our scheme requires choosing two (possibly overlapping) subsets of parties $\mathcal{P}_{(h)}$, $\mathcal{P}_{(1)} \subseteq \mathcal{P}$, such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ at least 1 honest party. To authenticate a secret value x, we first additively secret-share x between $\mathcal{P}_{(1)}$, and

[6] This can be chosen at random, or in some pre-agreed deterministic manner to load-balance communication among the parties.

then give every party in $\mathcal{P}_{(1)}$ a MAC on its share under a random MAC key given to each party in $\mathcal{P}_{(h)}$, as follows:

$$\begin{aligned} P_i \in \mathcal{P}_{(h)}: &\quad \Delta^i, \{\boldsymbol{k}^{i,j}[x^j]\}_{j\in\mathcal{P}_{(1)}, j\neq i} \\ P_j \in \mathcal{P}_{(1)}: &\quad x^j, \{\boldsymbol{m}^{j,i}[x^j]\}_{i\in\mathcal{P}_{(h)}, i\neq j} \\ \text{such that } x = \sum_j x^j &\quad \text{and} \quad \boldsymbol{m}^{j,i}[x^j] = \boldsymbol{k}^{i,j}[x^j] + x^j \cdot \Delta^i. \end{aligned}$$

where $\boldsymbol{k}^{i,j}[x^j]$ is a key chosen by P_i from $\{0,1\}^\ell$ to authenticate the message x^j that is chosen by P_j whereas $\boldsymbol{m}^{j,i}[x^j]$ is a MAC on a message x^j computed using the keys Δ^i and $\boldsymbol{k}^{i,j}[x^j]$. We denote this representation by $[x]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$. Note that sometimes we use representations with a different set of global keys $\Delta = \{\Delta^i\}_{i\in\mathcal{P}_{(h)}}$, but when it is clear from context we omit Δ and write $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$.

We remark that a special case is when $\mathcal{P}_{(h)} = \mathcal{P}_{(1)} = \mathcal{P}$, which gives the usual n-party representation of an additively shared value $x = x^1 + \cdots + x^n$, as used in [BDOZ11, BLN+15]:

$$[x] = \{x^i, \Delta^i, \{\boldsymbol{m}^{i,j}, \boldsymbol{k}^{i,j}\}_{j\neq i}\}_{i\in[n]}, \quad \boldsymbol{m}^{i,j} = \boldsymbol{k}^{j,i} + x^i \cdot \Delta^j,$$

where each party P_i holds the $n-1$ MACs $\{\boldsymbol{m}^{i,j}\}$ on x^i, as well as the keys $\boldsymbol{k}^{i,j}$ on each x^j, for $j \neq i$, and a global key Δ^i.

The idea behind our setup is that to cheat when opening x to all parties would require guessing at least h MAC keys of the honest parties in committee $\mathcal{P}_{(h)}$. In Figure 1 and Figure 2 we describe our protocols for opening values to a subset $\bar{\mathcal{P}} \subseteq \mathcal{P}$ and to a single party, respectively, and checking MACs. First each party in $\mathcal{P}_{(1)}$ broadcasts its share x^j to $\mathcal{P}_{(h)}$, and then later, when checking MACs, P_j sends the MAC $\boldsymbol{m}^{j,i}$ to P_i for verification. To improve efficiency, we make two optimizations to this basic method: firstly, instead of sending the individual MACs, when opening a large batch of values P_j only sends a single, random linear combination of all the MACs. Secondly, the verifier P_i does not check every MAC equation from each P_j, but instead sums up all the MACs and performs a single check. This has the effect that we only verify the sum x was opened correctly, and not the individual shares x^j.

Overall, to open x to an incorrect value x' requires guessing the Δ^i keys of all honest parties in $\mathcal{P}_{(h)}$, so can only be done with probability $\leq 2^{-h\ell}$. This means we can choose $\ell = \lambda/h$ to ensure security. Note that it is crucial when opening $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ that the shares x^j are *broadcast* to all parties in $\mathcal{P}_{(h)}$, to ensure consistency. Without this, a corrupt P_j could open, for example, an incorrect value to a single party in $\mathcal{P}_{(h)}$ with probability $2^{-\ell}$, and the correct share to all other parties.

More details on the correctness and security of our open and MACCheck protocols are given in the full version of this paper.

Efficiency Savings From Short Keys. Note that the reason for taking this approach is *not* to obtain a more efficient MAC scheme, but to design a scheme allowing more efficient *creation* of the MACs. Setting up the MACs typically requires oblivious transfer, with a communication cost proportional to the key length, so a smaller ℓ gives us direct

Protocol $\Pi_{[\mathsf{Open}]}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$

PARAMETERS: $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}, \bar{\mathcal{P}} \subseteq \mathcal{P}$ three (possibly overlapping) subsets of parties, such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party; $\ell =$ key length ; $m =$ number of authenticated bits

Open: To open $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ to a set of parties $\bar{\mathcal{P}} \subseteq \mathcal{P}$:

1. Each party $P_j \in \mathcal{P}_{(1)}$ broadcasts its share x^j to $\bar{\mathcal{P}}$.
2. All parties in $\bar{\mathcal{P}}$ locally compute $x = \sum_{j \in \mathcal{P}_{(1)}} x^j$.

Single Check: To check the MAC on the opened value x:

1. Each $P_j \in \mathcal{P}_{(1)}$ sends $\boldsymbol{m}^{j,i}[x]$ to every $P_i \in \mathcal{P}_{(h)} \setminus \{P_j\}$.
2. Each party in $\bar{\mathcal{P}}$ broadcasts the previously received x to $\mathcal{P}_{(h)}$.
3. Each $P_i \in \mathcal{P}_{(h)}$ checks that

$$\sum_{j \in \mathcal{P}_{(1)} \setminus P_i} \left(\boldsymbol{m}^{j,i}[x] + \boldsymbol{k}^{i,j}[x] \right) + (x + x^i) \cdot \Delta^i = 0$$

where $x^i = 0$ if $P_i \notin \mathcal{P}_{(1)}$. If any check fails, abort.

Batch Check: To check the MACs on a batch of opened values $x_1, \ldots, x_m$:

1. Parties in $\bar{\mathcal{P}}, \mathcal{P}_{(h)}, \mathcal{P}_{(1)}$ sample m random values $\chi_1, \ldots, \chi_m \leftarrow \mathcal{F}_{\mathsf{Rand}}(\mathbb{F}_{2^\lambda})$.
2. Each party in $\bar{\mathcal{P}}$ locally computes $\boldsymbol{y} = \sum_{k=1}^m \chi_k \cdot x_k \in \mathbb{F}_{2^\lambda}$ and broadcasts this to $\mathcal{P}_{(h)}$.
3. If parties in $\mathcal{P}_{(h)}$ receive inconsistent values, then abort.
4. Each $P_i \in \mathcal{P}_{(h)} \cap \mathcal{P}_{(1)}$ computes $\boldsymbol{y}^i = \sum_{k=1}^m \chi_k \cdot x_k^i$.
5. Parties in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ locally compute $[\boldsymbol{y}]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} = \sum_{k=1}^m \chi_k \cdot [x_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ (with multiplication over $\mathbb{F}_{2^\lambda}$).
6. Each $P_j \in \mathcal{P}_{(1)}$ sends $\boldsymbol{m}^{j,i}[\boldsymbol{y}]$ to every $P_i \in \mathcal{P}_{(h)} \setminus \{P_j\}$.
7. Each $P_i \in \mathcal{P}_{(h)}$ has received values $\{\boldsymbol{y}\}$ and MACs $\{\boldsymbol{m}^{j,i}[\boldsymbol{y}]\}, j \in \mathcal{P}_{(1)} \setminus \{P_i\}$, and checks that

$$\sum_{j \in \mathcal{P}_{(1)} \setminus \{P_i\}} \left(\boldsymbol{m}^{j,i}[\boldsymbol{y}] + \boldsymbol{k}^{i,j}[\boldsymbol{y}] \right) + (\boldsymbol{y} + \boldsymbol{y}^i) \cdot \Delta^i = 0,$$

where $\boldsymbol{y}^i = 0$ if $P_i \notin \mathcal{P}_{(1)}$. If any check fails, abort.

Fig. 1. Protocols for opening and MAC-checking on $(\mathcal{P}_{(h)}, \mathcal{P}_{(1)})$-authenticated secret shares

efficiency improvements to the preprocessing phase, which is by far the dominant cost in applications (see Section 5 for details). Regarding the scheme itself, notice that this is actually *less efficient*, in terms of storage and computation costs, than the distributed MAC scheme used in the SPDZ protocol [DKL+13], which only requires each party to store $\lambda + 1$ bits per authenticated Boolean value. However, it turns out that these overheads are less significant in practice compared with the communication cost of setting up the MACs, where we gain a lot.

Protocol $\Pi^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},P_{i_0}}_{\mathsf{PrivateOpen}}$

Private Open: To open a value $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ towards P_{i_0}:

1. Each $P_j \in \mathcal{P}_{(1)}$ sends their share x^j to P_{i_0}, who locally reconstructs $x = \sum_{j\in\mathcal{P}_{(1)}} x^j$.

Batch Check: To check the MACs on a batch of opened values $x_1, \ldots, x_m$:

1. Parties in $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$ call $\mathcal{F}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\mathsf{aBit}}$ to obtain λ random authenticated values $[r_1]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\Delta}, \ldots, [r_\lambda]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\Delta}$
2. Each $P_j \in \mathcal{P}_{(1)}$ sends $r^j_1, \ldots, r^j_\lambda$ to P_{i_0}.
3. Sample λ random values $\chi_k \leftarrow \mathbb{F}_{2^\lambda}$ using $\mathcal{F}_{\mathsf{Rand}}$, $k \in [m]$
4. P_{i_0} locally computes
$$\boldsymbol{y} = \sum_{k=1}^{m} \chi_k \cdot x_k + \sum_{k=1}^{\lambda} X^{k-1} \cdot r_k,$$
and broadcasts the result to $\mathcal{P}_{(h)}$.
5. Each $P_i \in \mathcal{P}_{(h)} \cap \mathcal{P}_{(1)}$ computes
$$\boldsymbol{y}^i = \sum_{k=1}^{m} \chi_k \cdot x^i_k + \sum_{k=1}^{\lambda} X^{k-1} \cdot r^i_k$$
6. Parties in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ locally compute
$$[\boldsymbol{y}]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\Delta} = \sum_{k=1}^{m} \chi_k \cdot [x_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\Delta} + \sum_{k=1}^{l} X^{k-1} \cdot [r_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\Delta},$$
where multiplication is performed over the finite field $\mathbb{F}_{2^\lambda}$.
7. Each $P_j \in \mathcal{P}_{(1)}$ privately sends $\boldsymbol{m}^{j,i}_{\Delta}[\boldsymbol{y}]$ to every $P_i \in \mathcal{P}_{(h)} \setminus P_j$.
8. Each $P_i \in \mathcal{P}_{(h)}$ has received MACs $\{\boldsymbol{m}^{j,i}_{\Delta}[\boldsymbol{y}]\}_{j\in\mathcal{P}_{(1)}\setminus P_i}$, and checks that
$$\sum_{j\in\mathcal{P}_{(1)}\setminus P_i} \left(\boldsymbol{m}^{j,i}_{\Delta}[\boldsymbol{y}] + \boldsymbol{k}^{i,j}_{\Delta}[\boldsymbol{y}]\right) + (\boldsymbol{y} + \boldsymbol{y}^i) \cdot \Delta^i = 0$$
where $\boldsymbol{y}^i = 0$ if $P_i \notin \mathcal{P}_{(1)}$. If any check fails, abort and notify all parties in $\mathcal{P}$.

Fig. 2. Protocol for privately opening $(\mathcal{P}_{(h)}, \mathcal{P}_{(1)})$-party authenticated secret shares to a single party P_{i_0} and MAC-checking

Extension to Arithmetic Shares. The scheme presented above can easily be extended to the arithmetic setting, with shares in a larger field instead of just $\mathbb{F}_2$. To do this with short keys, we simply choose the MAC keys Δ^i to be from a small subset of the field. For example, over $\mathbb{F}_p$ for a large prime p, each party chooses $\Delta^i \in \{0, \ldots, 2^\ell - 1\}$, and will obtain MACs of the form $m^{j,i} = k^{i,j} + x^j \cdot \Delta^i$ over $\mathbb{F}_p$, where $k^{i,j}$ is a random element of $\mathbb{F}_p$. This allows for a reduced preprocessing cost when generating MACs with the MASCOT protocol [KOS16] based on oblivious transfer: instead of requiring k OTs on k-bit strings between all $n(n-1)$ pairs of parties, where $k = \lceil \log_2 p \rceil$, we can adapt our preprocessing protocol from Section 5 to $\mathbb{F}_p$ so that we only need to perform

ℓ OTs on k-bit strings between $(n-1)(t+1)$ pairs of parties to set up each shared MAC.

3.1 Operations on $[\cdot]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$-Shared Values

Recall that $\mathcal{P}_{(h)} \cap \mathcal{P}_{(1)}$ is not necessarily the empty set.

Addition and multiplication with constant: We can define addition of $[x]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ with a public constant $c \in \{0,1\}$ by:

1. A designated $P_{i^*} \in \mathcal{P}_{(1)}$ replaces its share x^{i^*} with $x^{i^*} + c$.
2. Each P_i (for $i \in \mathcal{P}_{(h)}, i \neq i^*$) replaces its key $\boldsymbol{k}^{i,1}[x]$ with $\boldsymbol{k}^{i,1}[x] + c \cdot \Delta^i$. (All other values are unchanged.)

We also define multiplication of $[x]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ by a public constant $c \in \{0,1\}$ (or in $\{0,1\}^\ell$) by multiplying every share x^i, MAC $\boldsymbol{m}^{i,j}[x]$ and key $\boldsymbol{k}^{i,j}[x]$ by c.

Addition of shared values: Addition (XOR) of two shared values $[x]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, [y]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ is straightforward addition of the components. Note that it is possible to compute the sum $[x]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} + [y]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(h)}}$ of values shared within different committees in the same way, obtaining a $[x+y]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}}$ representation.

3.2 Converting to a More Compact Representation

We can greatly reduce the storage overhead in our scheme by "compressing" the MACs into a single, SPDZ-like sharing in *only* committee $\mathcal{P}_{(1)}$ with longer keys. Recall that the SPDZ protocol MAC representation [DPSZ12, DKL+13] of a secret bit x held by the parties in $\mathcal{P}_{(1)}$ is given by

$$[\![x]\!] = \{x^j, \boldsymbol{m}^j[x]\}_{j \in \mathcal{P}_{(1)}}$$

where each party P_j in $\mathcal{P}_{(1)}$ holds a share x^j, a MAC share $\boldsymbol{m}^j[x] \in \mathbb{F}_2^\lambda$ and a global MAC key share $\Delta^j \in \mathbb{F}_2^\lambda$, such that

$$x = \sum_{j \in \mathcal{P}_{(1)}} x^j, \quad \sum_{j \in \mathcal{P}_{(1)}} \boldsymbol{m}^j = (\sum_{j \in \mathcal{P}_{(1)}} x^j) \cdot (\sum_{j \in \mathcal{P}_{(1)}} \Delta^j)$$

Using this instead of the previous representation gives a much simpler and more efficient MAC scheme in the online phase of our MPC protocol, since each party only stores $\lambda + 1$ bits per value, instead of up to $|\mathcal{P}_{(h)}| \cdot \ell + 1$ bits with the scheme using short keys. Therefore, to obtain *both* the efficiency of *generating* MACs in the previous scheme, and *using* the MACs with SPDZ, below we show how to convert an inefficient, pairwise sharing $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ into a more compact SPDZ sharing $[\![x]\!]$. This procedure is shown in Figure 3.

Note that with the SPDZ representation, the parties in $\mathcal{P}_{(1)}$ can perform linear computations and openings (within $\mathcal{P}_{(1)}$) in just the same way. For completeness, we present the opening and MAC check protocols in the full version of this paper.

Protocol $\Pi_{\mathsf{MACCompact}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m,\ell}$

PARAMETERS: $\mathcal{P}_{(h)}, \mathcal{P}_{(1)} \subseteq \mathcal{P}$ two (possibly overlapping) subsets of parties, such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party; ℓ = key length; m = number of authenticated bits.
On input $[x_1]_{\Delta}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, \ldots, [x_m]_{\Delta}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$, do as follows:

1. Each $P_i \in \mathcal{P}_{(h)}$ samples and broadcasts $\boldsymbol{r}^i \leftarrow \mathbb{F}_{2^\lambda}$.
2. For $\iota \in [m]$:
   (a) Each $P_i \in \mathcal{P}_{(h)}$ computes $\boldsymbol{r}^i \cdot \sum_{j \in \mathcal{P}_{(1)}} \boldsymbol{k}^{i,j}[x_\iota]$, then reshares the resulting value to the parties in $P_j \in \mathcal{P}_{(1)}$, each of which obtains random shares $\{\tilde{\boldsymbol{k}}^{i,j}[x_\iota]\}_{i \in \mathcal{P}_{(h)}}$.
   (b) Each $P_i \in \mathcal{P}_{(h)}$ reshares $\Delta^i \cdot \boldsymbol{r}^i \in \mathbb{F}_{2^\lambda}$ to the parties in $\mathcal{P}_{(1)}$, each of which obtains $\{\tilde{\Delta}^{i,j}\}_{i \in \mathcal{P}_{(h)}}$.
   (c) Each $P_j \in \mathcal{P}_{(1)}$ outputs its part of $[\![x_\iota]\!]$ by computing the MAC share

$$\tilde{\boldsymbol{m}}^j[x_\iota] = \sum_{i \in \mathcal{P}_{(h)}} (\tilde{\boldsymbol{k}}^{i,j}[x_\iota] + \boldsymbol{m}^{j,i}[x_\iota] \cdot \boldsymbol{r}^i) \in \mathbb{F}_{2^\lambda}$$

   and key share $\tilde{\Delta}^j = \sum_{i \in \mathcal{P}_{(h)}} \tilde{\Delta}^{i,j} \in \mathbb{F}_{2^\lambda}$.

Fig. 3. Protocol for transforming $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ representations to $[\![x]\!]$ representations

To see correctness, first notice that from step 2a, we have that $\sum_j \tilde{k}^{i,j} = r^i \cdot \sum_j k^{i,j}$. So each party in $\mathcal{P}_{(1)}$ holds a share x^j and a MAC share $\tilde{\boldsymbol{m}}^j \in \mathbb{F}_{2^\lambda}$, which satisfy:

$$\begin{aligned}\sum_j \tilde{\boldsymbol{m}}^j &= \sum_j \sum_i (\tilde{\boldsymbol{k}}^{i,j} + \boldsymbol{m}^{j,i} \cdot r^i) \\ &= \sum_{i,j} (\boldsymbol{k}^{i,j} + \boldsymbol{m}^{j,i}) \cdot r^i = \sum_{i,j} x^j \cdot \Delta^i \cdot r^i = x \cdot \tilde{\Delta}.\end{aligned}$$

The security of this scheme now depends on the single, global MAC key $\tilde{\Delta} = \sum_i \Delta^i \cdot \boldsymbol{r}^i$, instead of the concatenation of Δ^i for $i \in \mathcal{P}_{(h)}$. Since at least h of the short keys $\Delta^i \in \mathbb{F}_{2^\ell}$ are unknown and uniformly random, from the leftover hash lemma [ILL89] it holds that $\tilde{\Delta}$ is within statistical distance $2^{-\lambda}$ of the uniform distribution over $\{0,1\}^\lambda$ as long as $h\ell \geq 3\lambda$. This gives a slightly worse bound than the previous scheme, but allows for a much more efficient *online phase* of the MPC protocol since, once the SPDZ representations are produced, only parties in $\mathcal{P}_{(1)}$ need to interact, and they have much lower storage and local computation costs. Note that in our instantiation of this scheme for the overall MPC protocol, we also need to choose the parameters h, ℓ such that the $\mathcal{L}$-DRSD assumption is hard; it turns out that all of our parameter choices (see Section 8) for this already satisfy $h\ell \geq 3\lambda$, so in this case using more compact MACs does not incur any extra overheads.

Improved Analysis for 1-bit Keys When the key length is 1, we can improve upon the previous bound from the leftover hash lemma with a more fine-grained analysis. Notice

that we can write the new key $\tilde{\Delta}$ as $\tilde{\Delta} = \mathbf{R} \cdot \Delta$, where $\mathbf{R} \in \{0,1\}^{\lambda \times n}$ is a matrix with $\boldsymbol{r}^i$ as columns. Since at least h positions of Δ are uniformly random, from randomness extraction results for bit-fixing sources (as used in, e.g. [NST17, Theorem 1]) it holds that since every honestly sampled row of $\mathbf{R}$ is uniformly random, $\tilde{\Delta}$ is within statistical distance $2^{\lambda - h}$ of the uniform distribution. We therefore require $h \geq 2\lambda$, instead of $h \geq 3\lambda$ as previously.

Optimization with Vandermonde Matrices Over Small Fields If we choose each of the Δ^i keys to come from a small finite field $\mathbb{F}$, with $|\mathbb{F}| \geq n$, then we can optimize the compact MAC scheme even further, so that there is *no overhead* on top of the previous pairwise scheme. The idea is to use a Vandermonde matrix to extract randomness from all parties' small MAC keys in a deterministic fashion, instead of using random vectors $\boldsymbol{r}^i$ as before. This technique is inspired by previous applications of hyper-invertible matrices to MPC in the honest majority setting [BTH08].

Let $v_1, \ldots, v_n$ be distinct points in $\mathbb{F}$, where $\mathbb{F}$ is such that $h \cdot |\mathbb{F}| \geq \lambda$. Now let $\mathbf{V} \in \mathbb{F}^{n \times h}$ be the Vandermonde matrix given by

$$\mathbf{V} = \begin{pmatrix} 1 & v_1 & \ldots & v_1^{h-1} \\ 1 & v_2 & \ldots & v_2^{h-1} \\ \vdots & \ddots & \ddots & \vdots \\ 1 & v_n & \ldots & v_n^{h-1} \end{pmatrix}$$

Party P_i defines the new MAC key share $\tilde{\Delta}^i = \boldsymbol{v}_i \cdot \Delta^i$, where $\boldsymbol{v}_i$ is the i-th row of $\mathbf{V}$. This results in a new global key given by $\tilde{\Delta} = (\Delta^1, \ldots, \Delta^n) \cdot \mathbf{V} \in \mathbb{F}^h$. From the fact that at least h components of Δ are uniformly random, and the property of the Vandermonde matrix that any square matrix formed by taking h rows of $\mathbf{V}$ is invertible, it follows that $\tilde{\Delta}$ is a uniformly random vector in $\mathbb{F}^h$. More formally, this means that if $n - h$ components of Δ are fixed and we define Δ_H to be the h honest MAC key components, then the mapping $\Delta_H \mapsto \Delta \cdot \mathbf{V}$ is a bijection, so $\tilde{\Delta}$ is uniformly random as long as Δ_H is. Therefore we can choose $h \geq \lambda / |\mathbb{F}|$ to obtain $\leq 2^{-\lambda}$ cheating probability in the resulting MAC scheme.

Allowing leakage on the MAC keys. In our subsequent protocol for generating MACs, to obtain an efficient protocol we need to allow some *leakage* on the individual MAC keys $\Delta^i \in \{0,1\}^\ell$, in the form of allowing the adversary to guess a single bit of information on each Δ^i. For both the pairwise MAC scheme and the compact, SPDZ-style MACs, this leakage does not affect an adversary's probability of forging MACs in our actual protocols, since the entire MAC key still needs to be guessed to break security — allowing guesses on smaller parts of the key does not help, as a single incorrect guess causes the protocol to abort. We analyse the security of this for our compact MAC representation in the full version.

4 Correlated OT on Short Strings

As a building block, we need a protocol for random correlated oblivious transfer (or random Δ-OT) on short strings. This is a 2-party protocol, where the receiver inputs bits $x_1, \ldots, x_m$, the sender inputs a short string $\Delta \in \{0,1\}^\ell$, and the receiver obtains random strings $\boldsymbol{t}_i \in \{0,1\}^\ell$, while the sender learns $\boldsymbol{q}_i = \boldsymbol{t}_i + x_i \cdot \Delta$. The ideal functionality for this is shown in Figure 4.

The protocol we use to realise this (shown in in the full version of this paper) is a variant of the OT extension protocol of Keller et al. [KOS15], modified to produce correlated OTs (as done in [NST17]) and with short strings. The security of the protocol can be shown similarly to the analysis of [KOS15]. That work showed that a corrupt party may attempt to guess a few bits of information about the sender's secret Δ, and will succeed with probability 2^{-c}, where c is the number of bits. In our case, since Δ is small, a corrupt receiver may actually guess *all of* Δ with some noticeable probability, in which case all security for the sender is lost. This is modelled in the functionality $\mathcal{F}_{\Delta\text{-ROT}}$, which allows a corrupt receiver to submit such a guess. This leakage does not cause a problem in our multi-party protocols, because an adversary would have to guess the keys of *all* honest parties to break security, and this can only occur with negligible probability.

Communication complexity. Recall that λ is the statistical security parameter and κ the computational security parameter. The initialization phase requires ℓ random OTs, which costs $\ell\kappa$ bits of communication when implemented using OT extension. The communication complexity of the **Extend** phase, to create m Δ-ROTs, is $\ell(m+\lambda)$ bits to create the OTs, and $\kappa + 2\lambda$ bits for the consistency check (we assume P_S only sends a κ-bit seed used to generate the χ_i's). This gives an amortized cost of $\ell + (\kappa + 3\lambda)/m$ bits per Δ-ROT, which is less than $\ell + 4$ bits when $m > \kappa$.

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

Initialize: Upon receiving (Init, Δ), where $\Delta \in \{0,1\}^\ell$ from P_S and (Init) from P_R, store Δ. Ignore any subsequent (Init) commands.
Extend: Upon receiving $(\mathsf{extend}, x_1, \ldots, x_m)$ from P_R, where $x_i \in \{0,1\}$, and (extend) from P_S, do the following:

- Sample $\boldsymbol{t}_i \in \{0,1\}^\ell$, for $i \in [m]$. If P_R is corrupted then wait for $\mathcal{A}$ to input $\boldsymbol{t}_i$.
- Compute $\boldsymbol{q}_i = \boldsymbol{t}_i + x_i \cdot \Delta$, for $i \in [m]$.
- If P_S is corrupted then wait for $\mathcal{A}$ to input $\boldsymbol{q}_i \in \{0,1\}^\ell$ and recompute $\boldsymbol{t}_i = \boldsymbol{q}_i + x_i \cdot \Delta$.
- Output $\boldsymbol{t}_i$ to P_R and $\boldsymbol{q}_i$ to P_S, for $i \in [m]$.

Key queries: If P_R is corrupt then on receiving an efficiently computable predicate $P : \{0,1\}^\ell \to \{0,1\}$ from the adversary, send 1 to the adversary if $P(\Delta) = 1$. If $P(\Delta) = 0$ then the functionality aborts.

Fig. 4. Functionality for oblivious transfer on random, correlated strings.

5 Bit Authentication with Short Keys

Functionality $\mathcal{F}_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m,\ell}$

PARAMETERS: m number of authenticated bits; ℓ key length.

Initialize: On receiving $(\mathsf{Init}, \Delta^i)$ from $P_i \in \mathcal{P}_{(h)}$ and (Init) from $P_j \in \mathcal{P}_{(1)}$, store all $\Delta^i \in \{0,1\}^\ell$.
aBit: On receiving (aBit, m) from $P_i \in \mathcal{P}_{(h)}$ and $(\mathsf{aBit}, \boldsymbol{x}^j = (x_1^j, \ldots, x_m^j))$ from every $P_j \in \mathcal{P}_{(1)}$:

1. Run $(\mathcal{P}_{(h)}, \mathcal{P}_{(1)})$-$\mathsf{Bracket}(\{x_h^j\}_{j\in\mathcal{P}_{(1)}})$ (see below), for every $h \in [m]$.
2. Output $\Delta^i, \{\boldsymbol{k}^{i,j}[x_h^j]\}_{j\in\mathcal{P}_{(1)}\setminus\{i\}}$ to every $P_i \in \mathcal{P}_{(h)}$ and $x_h^j, \{\boldsymbol{m}^{j,i}[x_h^j]\}_{i\in\mathcal{P}_{(h)}\setminus\{j\}}$ to every $P_j \in \mathcal{P}_{(1)}$.

Key queries: Upon receiving (i, P) from the adversary, where P is an efficiently computable predicate $P : \{0,1\}^\ell \to \{0,1\}$ and $i \in \mathcal{P}_{(h)}$, output 1 to the adversary if $P(\Delta^i) = 1$. Otherwise, abort.

Fig. 5. Functionality for authenticated bits

In this section we describe our protocols for authenticating bits with short MAC keys. To capture the short keys used for authentication we need to define a series of different functionalities.

5.1 Authenticated Bit Functionality $\mathcal{F}_{\mathsf{aBit}}$

We begin with the description of the ideal functionality $\mathcal{F}_{\mathsf{aBit}}$ described in Figure 5 that formalises the MACs we create. Each party $P_i \in \mathcal{P}_{(h)}$ chooses a global $\Delta^i \in \{0,1\}^\ell$, then $\mathcal{F}_{\mathsf{aBit}}$ calls the subroutine $(\mathcal{P}_{(h)}, \mathcal{P}_{(1)})$-$\mathsf{Bracket}$ (Figure 6) that uses these global MAC keys $\{\Delta^i\}_{i\in\mathcal{P}_{(h)}}$ stored by the functionality to create pairwise MACs of the same length, as illustrated in Section 3.

5.2 Bit Authentication Protocol

We now present our bit authentication protocol Π_{aBit}, described in Figure 7, implementing the functionality $\mathcal{F}_{\mathsf{aBit}}$ (Figure 5). The protocol first runs the Δ-OT protocol with short keys between every pair of parties in $\mathcal{P}_{(h)} \times \mathcal{P}_{(1)}$ to authenticate the additively shared inputs, in a standard manner. We then need to adapt the consistency check from the TinyOT-style authentication protocol presented by Hazay et al. ([HSS17]) to our setting of MACs with short keys distributed between two committees, to ensure that all parties input consistent values in all the COT instances.

Taking a closer look at the consistency checks in Step 3f, the first check verifies the consistency of the Δ^i values, whereas in the second set of checks we test the consistency

Macro $(\mathcal{P}_{(h)}, \mathcal{P}_{(1)})$-Bracket

On input $\{x^j\}_{j\in\mathcal{P}_{(1)}}$, authenticate the share $x^j \in \{0,1\}$, for each $j \in \mathcal{P}_{(1)}$, as follows:
P_j CORRUPT: Receive a MAC $\boldsymbol{m}^{j,i} \in \mathbb{F}_2^\ell$ from $\mathcal{A}$ and compute the keys $\{\boldsymbol{k}^{i,j} = \boldsymbol{m}^{j,i} + x^j \cdot \Delta^i\}_{i\in(\mathcal{P}_{(h)}\setminus\{P_j\})}$.
OTHERWISE:

1. Sample honest parties' keys $\boldsymbol{k}^{i,j} \leftarrow \mathbb{F}_2^\ell$, for $P_i \in \mathcal{P}_{(h)} \setminus (A \cup \{P_j\})$.
2. Receive keys $\boldsymbol{k}^{i,j} \in \mathbb{F}_2^\ell$, for each $P_i \in A \cap \mathcal{P}_{(h)}$, from the adversary.
3. Compute the MACs $\boldsymbol{m}^{j,i} = \boldsymbol{k}^{i,j} + x^j \cdot \Delta^i$, where $P_i \in \mathcal{P}_{(h)}$.

Output $[x]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$.

Fig. 6. Macro used by $\mathcal{F}_{\mathsf{aBit}}$ to authenticate bits

of the individual shares x^j. To see correctness when all parties are honest, notice that in the first check, for $i \in \mathcal{P}_{(h)}$ we have:

$$\begin{aligned}
&\boldsymbol{z}^i + \sum_{j\in(\mathcal{P}_{(1)}\setminus\{P_i\})} \boldsymbol{z}^{j,i} = 0 \\
&\iff (\boldsymbol{y}^i + \boldsymbol{y}) \cdot \Delta^i + \sum_{j\in(\mathcal{P}_{(1)}\setminus\{P_i\})} (\boldsymbol{k}^{i,j}[\boldsymbol{y}] + \boldsymbol{m}^{j,i}[\boldsymbol{y}]) = 0 \\
&\iff (\boldsymbol{y}^i + \boldsymbol{y}) \cdot \Delta^i + \sum_{j\in(\mathcal{P}_{(1)}\setminus\{P_i\})} (\boldsymbol{y}^j \cdot \Delta^i) = 0 \iff \boldsymbol{y} \cdot \Delta^i + \boldsymbol{y} \cdot \Delta^i = 0.
\end{aligned}$$

For a corrupt party who misbehaves during the protocol, there are two potential deviations:

1. A corrupt $P_i, i \in \mathcal{P}_{(h)}^{\mathcal{A}}$ provides an inconsistent $\Delta^{i,j}$ when acting as a sender in $\mathcal{F}_{\Delta\text{-ROT}}^{m,\ell}$ with different honest parties, i.e. $\Delta^i \neq \Delta^{i,j}$ for some $j \in \mathcal{P}_{(1)} \setminus A$.
2. A corrupt $P_j, j \in \mathcal{P}_{(1)}^{\mathcal{A}}$ provides an inconsistent input $x_\iota^{i,j}$ when acting as a receiver in $\mathcal{F}_{\Delta\text{-ROT}}^{m,\ell}$ with different parties, i.e. $x_\iota^i \neq x_\iota^{i,j}$, for some $j \in \mathcal{P}_{(h)} \setminus A$.

Note that in the above, the 'correct' inputs Δ^i, x_ι^j for a corrupt $P_i \in \mathcal{P}_{(h)}$ or $P_j \in \mathcal{P}_{(1)}$ are defined to be those in the $\mathcal{F}_{\Delta\text{-ROT}}$ instance with some fixed, honest party $P_{i_1} \in \mathcal{P}_{(1)}$ or $P_{j_1} \in \mathcal{P}_{(h)}$, respectively. We now prove the following two claims.

Claim 5.1 *Assuming a non-abort execution, then for every corrupted party $P_i, i \in \mathcal{P}_{(h)}^{\mathcal{A}}$, all Δ^i are consistent.*

Proof: In order to ensure that all Δ^i are consistent we use the first check. More precisely, we fix $P_j \in \mathcal{P}_{(h)}^{\mathcal{A}}$ and check that $\sum_{i\in[n]} \boldsymbol{z}^{i,j} = 0, \forall j$. Since we require that $\boldsymbol{y} \in \{0,1\}^\lambda$, the probability to pass the check is $1/2^\lambda$. More formally, let us assume that a corrupt P_j^* uses inconsistent $\Delta^{j,i}$ in $\mathcal{F}_{\Delta\text{-ROT}}$ with some $i \notin \mathcal{P}_{(h)}^{\mathcal{A}}$, then to pass the check P_j^* can send adversarial values in step 3c, i.e. when it broadcasts values $\bar{\boldsymbol{y}}^j$,

or in step 3d, when committing to the values $\boldsymbol{z}^{j,i}$. Let $\boldsymbol{e}_y \in \{0,1\}^\lambda$ denote an additive error so that $\sum_{i\in[n]} \bar{\boldsymbol{y}}^i = \boldsymbol{y} + \boldsymbol{e}_y$, and let $\boldsymbol{e}_z \in \{0,1\}^\lambda$ denote an additive error so that $\sum_{j\in\mathcal{P}^{\mathcal{A}}_{(h)}} \hat{\boldsymbol{z}}^{j,i} = \sum_{j\in\mathcal{P}^{\mathcal{A}}_{(h)}} \boldsymbol{z}^{j,i} + \boldsymbol{e}_z$. Finally, let $\delta^{j,i} = \Delta^j + \Delta^{j,i}$. Then if the check passes, it holds that:

$$0 = \sum_i \boldsymbol{z}^{i,j} = \boldsymbol{e}_z + \boldsymbol{z}^j + \sum_{i\neq j} \boldsymbol{z}^{i,j} = \boldsymbol{e}_z + (\boldsymbol{y} + \boldsymbol{e}_y + \boldsymbol{y}^j)\cdot\Delta^j + \sum_{i\neq j} \boldsymbol{y}^i\cdot\Delta^{j,i}$$
$$\iff \quad \boldsymbol{e}_z + \boldsymbol{e}_y\cdot\Delta^j = \sum_{i\neq j} \boldsymbol{y}^i\cdot\delta^{j,i},$$

which implies that the additive errors $\boldsymbol{e}_z$ and $\boldsymbol{e}_y$, that make the above equation equal to zero, depend on the $\boldsymbol{y}^i$ values, and that the adversary has to guess at least one of them in order to pass the check. This event happens with probability $2^{-\lambda}$ since the only information the adversary has about these values is that they are uniform additive shares of $\boldsymbol{y}$, due to the randomization in step 3c. □

Claim 5.2 *Assuming a non-abort execution, then for every corrupted party $P_j, j \in \mathcal{P}^{\mathcal{A}}_{(1)}$, all $x^{i,j}_\iota$ are consistent.*

Proof: We need to check that a corrupt P^*_j cannot input inconsistent $x^{j,i}$ to different honest parties without being caught. For every ordered pair of parties (P_i, P_j), we can define P_j's MAC $\boldsymbol{m}^{j,i}[\boldsymbol{y}]$ and P_i's key $\boldsymbol{k}^{i,j}[\boldsymbol{y}]$ respectively as

$$\sum_{\iota=1}^{m} \chi_\iota\cdot\boldsymbol{m}^{j,i}[x_\iota] + \sum_{k=1}^{\lambda} X^{k-1}\cdot\boldsymbol{m}^{j,i}[r_k] \quad \text{and}$$
$$\sum_{\iota=1}^{m} \chi_\iota\cdot\boldsymbol{k}^{i,j}[x_\iota] + \sum_{k=1}^{\lambda} X^{k-1}\cdot\boldsymbol{k}^{i,j}[r_k]\ .$$

A corrupt P_j can commit to incorrect MACs $\hat{\boldsymbol{z}}^{j,i}$, so that $\hat{\boldsymbol{z}}^{j,i} = \boldsymbol{z}^{j,i} + \boldsymbol{e}^{j,i}_z$ and $\hat{\boldsymbol{y}}^j = \boldsymbol{y}^{j,i} + \boldsymbol{e}^{j,i}_y$. In order to have the check passed, we have:

$$\boldsymbol{z}^{j,i} + \boldsymbol{e}^{j,i}_z = \boldsymbol{k}[\boldsymbol{y}]^{i,j} + (\boldsymbol{y}^{j,i} + \boldsymbol{e}^j_y)\cdot\Delta^i,$$

Which happens if and only if:

$$\boldsymbol{e}^{j,i}_z + (\boldsymbol{y}^{j,i} + \boldsymbol{e}^j_y)\cdot\Delta^i = \boldsymbol{m}^{j,i}[\boldsymbol{y}] + \boldsymbol{k}^{i,j}[\boldsymbol{y}]$$
$$= \big(\sum_{\iota=1}^{m} \chi_\iota\cdot(x^j_\iota + \delta^{j,i}_\iota) + \sum_{k=1}^{\lambda} X^{k-1}\cdot(r^j_k + \delta'^{j,i}_k)\big)\cdot\Delta^i$$
$$\iff \boldsymbol{e}^{j,i}_z = \big(\boldsymbol{y}^{j,i} + \boldsymbol{e}^j_y + \sum_{\iota=1}^{m} \chi_\iota\cdot(x^j_\iota + \delta^{j,i}_\iota) + \sum_{k=1}^{\lambda} X^{k-1}\cdot(r^j_k + \delta'^{j,i}_k)\big)\cdot\Delta^i$$
$$= (\boldsymbol{e}^j_y + \sum_{\iota=1}^{m} \chi_\iota\cdot\delta^{j,i}_\iota + \sum_{k=1}^{\lambda} X^{k-1}\cdot\delta'^{j,i}_k)\cdot\Delta^i.$$

Then there are two cases for which the adversary can pass the check:

1. In case $\boldsymbol{e}_z^{j,i} = (\boldsymbol{e}_y^j + \sum_{\iota=1}^{m} \chi_\iota \cdot \delta_\iota^{j,i} + \sum_{k=1}^{\lambda} X^{k-1} \cdot \delta_k'^{j,i}) \cdot \Delta^i \neq 0$ the adversary needs to guess Δ^i, which can only happen with probability $2^{-\ell}$. Note that in order to pass this check the adversary needs to guess all honest parties' keys. This is due to the fact that a corrupted P_j opens the same $\hat{\boldsymbol{y}}^j$ to all parties, so if it cheats and provides an inconsistent value then it must pass the above check with respect to all honest parties. Therefore, the overall probability of passing this check is $2^{-\ell h} \leq 2^{-\lambda}$.
2. In case $\boldsymbol{e}_z^{j,i} = 0$ and $\boldsymbol{e}_y^j = \sum_{\iota=1}^{m} \chi_\iota \cdot \delta_\iota^{j,i} + \sum_{k=1}^{\lambda} X^{k-1} \cdot \delta_k'^{j,i}, \forall i \notin \mathcal{P}_{(1)}^{\mathcal{A}}$. Assuming that there is at least one $i \notin \mathcal{P}_{(h)}^{\mathcal{A}}$ s.t. $\delta_\iota^{j,i} = \delta^i = 0$ (recall that we view the inputs of P_j in the interaction with party P_{j_1} as the 'correct' inputs, then there must be at least one party for which this condition holds). This implies that $\boldsymbol{e}_y^j = 0$ as well. Thus, for every $i \notin \mathcal{P}_{(h)}^{\mathcal{A}} \cup j_1$ it needs to holds that

$$0 = \sum_{\iota=1}^{m} \chi_\iota \cdot \delta_\iota^{j,i} + \sum_{k=1}^{\lambda} X^{k-1} \cdot \delta_k'^{j,i}.$$

   Since each χ_ι is uniformly random in $\mathbb{F}_{2^\lambda}$ and independent of the $\delta_\iota^{j,i}, \delta_\iota'^{j,i}$ values, it is easy to see that this only holds with probability $2^{-\lambda}$ if any $\delta_\iota^{j,i}$ is non-zero.

□

In the full version we prove the following theorem.

Theorem 5.1 *Protocol* $\Pi_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m,\ell}$ *securely implements the functionality* $\mathcal{F}_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m,\ell}$ *in the* $(\mathcal{F}_{\Delta\text{-}\mathsf{ROT}}^{m,\ell}, \mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{Commit}})$*-hybrid model.*

5.3 Efficiency Analysis

We now analyse the efficiency of our protocol and compare it with the previous best known approach to secret-shared bit authentication. When there are n parties with h honest, the previous best approach would be to use the standard TinyOT-style MAC scheme (as in [WRK17b, HSS17]) inside a committee of size $n - h + 1$ parties, to guarantee at least one honest party. Here, the MACs must be of length at least λ, and the amortized communication complexity can be around $\lambda(n - h + 1)(n - h)$ bits per authenticated bit. In contrast, in our scheme we have two committees of sizes n_1 and n_2, with h and 1 honest party, respectively. If we suppose the committees are deterministically chosen from a set of n parties with h honest, then we get $n_1 = n$ and $n_2 = n - h + 1$. To ensure security of the MAC scheme we need MACs of length $\ell \geq \lambda/h$, for statistical security λ. This gives an amortized complexity for creating a MAC of around $\ell n_1 n_2 = \lambda n(n - h + 1)/h$ bits. Compared with the TinyOT approach, this gives a reduction in communication of $h(n - h)/n$ times in our protocol. This is maximized when $h = n/2$, with a $n/4$ times reduction in communication cost over TinyOT, and for smaller h we still have savings for all $h > 1$.

Protocol $\Pi_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m,\ell}$

PARAMETERS: m, number of authenticated bits; ℓ, key length.

Initialize: On input Δ^i from each $P_i \in \mathcal{P}_{(h)}$, every pair of parties $(P_i, P_j) \in \mathcal{P}_{(h)} \times \mathcal{P}_{(1)},\ i \neq j$ calls $\mathcal{F}_{\Delta\text{-}\mathsf{ROT}}^{\ell}$ functionality with input $(\mathsf{Init}, \Delta^i)$ from P_i and (Init) from P_j.
aBit: On input $(x_1^j, \ldots, x_m^j) \in \{0,1\}^m$ from each $P_j \in \mathcal{P}_{(1)}$:

1. Each $P_j \in \mathcal{P}_{(1)}$ samples λ random bits $r_k^j, k \in [\lambda]$.
2. Call $\mathcal{F}_{\Delta\text{-}\mathsf{ROT}}^{\ell}$ with inputs (x_ι^j, r_k^j) from P_j, so P_j gets $\boldsymbol{t}^{j,i}$ and P_i gets $\boldsymbol{q}^{i,j}$, such that for $\iota \in [m], k \in [\lambda]$,

$$\boldsymbol{t}_\iota^{j,i} + \boldsymbol{q}_\iota^{i,j} = x_\iota^j \cdot \Delta^i \qquad \boldsymbol{t}_k^{j,i} + \boldsymbol{q}_k^{i,j} = r_k^j \cdot \Delta^i$$

   For $\iota \in [m]$, define $[x_\iota]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ as follows.
   Each $P_i \in \mathcal{P}_{(h)}$ sets $\boldsymbol{k}^{i,j}[x_\iota] = \boldsymbol{q}_\iota^{i,j}$ for $j \in (\mathcal{P}_{(1)} \setminus \{P_i\})$ and each $P_j \in \mathcal{P}_{(1)}$ sets $\boldsymbol{m}^{j,i}[x_\iota] = \boldsymbol{t}_\iota^{j,i}$ for $i \in (\mathcal{P}_{(h)} \setminus \{P_j\})$.
   For $k \in [s]$, the parties define $[r_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$: Each $P_i \in \mathcal{P}_{(h)}$ sets $\boldsymbol{k}^{i,j}[r_k] = \boldsymbol{q}_k^{i,j}$ for $j \in (\mathcal{P}_{(1)} \setminus \{P_i\})$ and each $P_j \in \mathcal{P}_{(1)}$ sets $\boldsymbol{m}^{j,i}[r_k] = \boldsymbol{t}_k^{j,i}$ for $i \in (\mathcal{P}_{(h)} \setminus \{P_j\})$.
3. Check consistency of the inputs as follows:
   (a) Call $\mathcal{F}_{\mathsf{Rand}}$ to obtain m field elements $\chi_\iota \in \mathbb{F}_2^\lambda, \iota \in [m]$.
   (b) Locally compute, over $\mathbb{F}_{2^\lambda}$, the shares

$$[\boldsymbol{y}]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} = \sum_{\iota=1}^{m} \chi_\iota \cdot [x_\iota]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} + \sum_{k=1}^{\lambda} X^{k-1} \cdot [r_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} ,$$

   so that each $P_j \in \mathcal{P}_{(1)}$ holds a share $\boldsymbol{y}^j \in \mathbb{F}_2^\lambda$, and MACs $\{\boldsymbol{m}^{j,i}[\boldsymbol{y}]\}_{i \in \mathcal{P}_{(h)} \setminus P_j}$ and each $P_i \in \mathcal{P}_{(h)}$ holds keys $\{\boldsymbol{k}^{i,j}[\boldsymbol{y}]\}_{j \in \mathcal{P}_{(1)} \setminus P_i}$.
   (c) The parties in $\mathcal{P}_{(1)}$ call $\mathcal{F}_{\mathsf{Zero}}^{\ell}$ so that each $P_j \in \mathcal{P}_{(1)}$ obtains a zero-share $\boldsymbol{\rho}^j \in \{0,1\}^\ell$. P_j then broadcasts $\bar{\boldsymbol{y}}^j := \boldsymbol{y}^j + \boldsymbol{\rho}^j$, and reconstructs $\boldsymbol{y} = \sum_{j \in \mathcal{P}_{(1)}} \bar{\boldsymbol{y}}^j$.
   (d) Each $P_i \in \mathcal{P}_{(h)}$ defines and commits, to all parties in $\mathcal{P}_{(h)}$, the following values. Note that $\boldsymbol{y}^i = 0$ if $P_i \notin \mathcal{P}_{(1)}$:

$$\boldsymbol{z}^i = (\boldsymbol{y}^i + \boldsymbol{y}) \cdot \Delta^i + \sum_{j \in \mathcal{P}_{(1)} \setminus P_i} \boldsymbol{k}^{i,j}[\boldsymbol{y}].$$

   (e) Each $P_j \in \mathcal{P}_{(1)}$ defines and commits, to all parties in $\mathcal{P}_{(h)}$:

$$\boldsymbol{y}^j, \quad \{\boldsymbol{z}^{j,i} = \boldsymbol{m}^{j,i}[\boldsymbol{y}]\}_{i \in \mathcal{P}_{(h)} \setminus P_j}.$$

   (f) Each party in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ opens its commitments and parties in $\mathcal{P}_{(h)}$ check that:

$$\forall i \in \mathcal{P}_{(h)}, \quad \boldsymbol{z}^i + \sum_{j \in \mathcal{P}_{(1)} \setminus P_i} \boldsymbol{z}^{j,i} = 0$$

   and

$$\forall j \in \mathcal{P}_{(1)}, i \in \mathcal{P}_{(h)} \setminus P_j, \quad \boldsymbol{z}^{j,i} = \boldsymbol{k}^{i,j}[\boldsymbol{y}] + \boldsymbol{y}^j \cdot \Delta^i.$$

   If any of these checks fails, abort.
4. Output $[x_1], \ldots, [x_m]$.

Fig. 7. Protocol for authentication of random shared bits using committees

6 Actively Secure MPC Protocol with Short Keys

Similarly to prior constructions such as [DPSZ12, NNOB12, FKOS15, KOS16], our protocol is in the pre-processing model where the main difference is that the computation is carried out via two random committees $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$. The preprocessing phase is function and input independent, and provides all the correlated randomness needed for the online phase where the function is securely evaluated .

6.1 The Online Phase

Our online protocol, shown in Figure 8, runs mostly as that of [DPSZ12, DKL+13] within a small committee $\mathcal{P}_{(1)} \subseteq \mathcal{P}$ with at least 1 honest party. The main difference is that we need the help of the bigger $\mathcal{P}_{(h)} \subseteq \mathcal{P}$ committee with at least h honest parties to authenticate the inputs of any $P_i \in \mathcal{P}$ using the $[\cdot]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$-representation before converting them to the more compact $[\![\cdot]\!]$-representation described in Section 3.2.

The Boolean MPC Protocol - Π_{BBB}

Prep: Parties call $\mathcal{F}_{\mathsf{Preprocessing}}$ with (Prep, m, M) to generate m random $[\cdot]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ bits and M random multiplication triples with compact MACs.

Input: To authenticate an input x of $P_i \in \mathcal{P}$:

1. Call the **Private Open** command in $\Pi^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},P_i}_{\mathsf{PrivateOpen}}$ on an unused bit $([r]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, P_i)$ from **Prep**, so only P_i learns r.
2. Call the **Batch Check** command in $\Pi^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},P_i}_{\mathsf{PrivateOpen}}$ on all the privately opened values in the previous step. If the check fails, abort.
3. P_i broadcasts $d = x + r$ to $\mathcal{P}_{(1)}$, who compute $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} = [r]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}} + d$.
4. Call $\Pi_{\mathsf{MACCompact}}$ on input $[x]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ to obtain $[\![x]\!]$.

Add: On input $([\![x]\!], [\![y]\!])$, locally compute $[\![x + y]\!] = [\![x]\!] + [\![y]\!]$.

Multiply: On input $[\![x]\!], [\![y]\!]$, parties in $\mathcal{P}_{(1)}$ do the following:

1. Pick an unused random multiplicative triple from **Prep** $([\![a]\!], [\![b]\!], [\![c]\!])$.
2. Compute $[\![\epsilon]\!] = [\![x]\!] + [\![a]\!]$, and $[\![\rho]\!] = [\![y]\!] + [\![b]\!]$ and call $\Pi_{[\![\mathsf{Open}]\!]}$ on these to reveal ϵ, ρ to $\mathcal{P}_{(1)}$.
3. Parties set $[\![x \cdot y]\!] = [\![c]\!] + \epsilon \cdot [\![b]\!] + \rho \cdot [\![a]\!] + \epsilon \cdot \rho$.

Output: To output a value $[\![x]\!]$ to $\mathcal{P}$ or a subset of it, do the following:

1. Call **BatchCheck** on $\Pi_{[\![\mathsf{Open}]\!]}$ for all $[\![\cdot]\!]$ values opened so far. If the check fails, abort.
2. Call commands **Open** and **Single Check** of $\Pi_{[\![\mathsf{Open}]\!]}$ on input $[\![x]\!]$. If the check fails, abort, otherwise accept x as a valid output.

Fig. 8. The Boolean MPC Protocol

6.2 The preprocessing Phase

The task of $\mathcal{F}_{\mathsf{Preprocessing}}$ is to create random authenticated bits under the $[\cdot]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$-representation and random authenticated triples under the compact $[\![\cdot]\!]$-representation.

7 Triple Generation

Here we present our triple generation protocol implementing the functionality described in Figure 9. First, protocol $\Pi_{\mathsf{HalfAuthTriple}}$ (Figure 11) implements the functionality $\mathcal{F}_{\mathsf{HalfAuthTriple}}$ (Figure 10) to compute cross terms in triples: each party $P_i \in \mathcal{P}_{(h)}$ inputs random shares $y_k^i, k \in [m]$, and committees $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$ obtain random representations $[x_k]_\Delta$ as well as shares of the cross terms defined by $\sum_{i\in\mathcal{P}_{(h)}} \sum_{j\neq\mathcal{P}_{(1)}\setminus\{P_i\}} x_k^j \cdot y_k^i, k \in [m]$.

Given this intermediate functionality, protocol Π_{Triple} (Figure 12) implements $\mathcal{F}_{\mathsf{Triple}}^{m,\ell}$ (Figure 9) computing correct authenticated and non-leaky triples $(\llbracket x_k \rrbracket, \llbracket y_k \rrbracket, \llbracket z_k \rrbracket)$ such that $(\sum_{j\in\mathcal{P}_{(1)}} x_k^j) \cdot (\sum_{j\in\mathcal{P}_{(1)}} y_k^j) = \sum_{j\in\mathcal{P}_{(1)}} z_k^j$. Checking correctness and removing leakage is achieved using classic cut-and-choose and bucketing techniques. Note that even though the final triples are under the compact $\llbracket\cdot\rrbracket$-representation we produce them first using $[\cdot]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$-representations in order to *generate* MACs more efficiently and having an efficient implementation of $\mathcal{F}_{\mathsf{HalfAuthTriple}}$.

It is crucial to note that the security of $\Pi_{\mathsf{HalfAuthTriple}}$ is based on the hardness of RSD, and for this reason the number of triples r generated by this protocol depends on the security RSD. So while essentially an unlimited number of random correlated OTs and random authenticated bits can be produced as described on previous sections, a naive use of short keys would actually result in an upper bound on the number of triples that can be produced securely. To fix this issue, during $\Pi_{\mathsf{HalfAuthTriple}}$ we make the parties 'switch the correlation' on representations $[x]_\Delta$, so they output a new representation under an independent correlation $[x]_{\tilde{\Delta}}$, with $\Delta \neq \tilde{\Delta}$ being the relevant value for the RSD assumption. Finally, the fact that $\tilde{\Delta}$ is short combined with the adversarial possibility of querying some predicates about it requires the reduction to use an interactive version of RSD, which we denote by $\mathcal{L}$-DRSD as in Definition 2.1.

Functionality $\mathcal{F}_{\mathsf{Triple}}^{m,\ell}$

PARAMETERS: m, number of multiplications; ℓ, key length.
This functionality runs in committee $\mathcal{P}_{(1)}$ only.

On input $(\mathsf{Triples}, m, \ell)$ from all parties, generate m random authenticated triples as follows.
Initialize: Receive Δ^i from the adversary for each corrupt $P_i \in \mathcal{P}_{(1)}^{\mathcal{A}}$ and sample $\Delta^i \leftarrow \mathbb{F}_2^\lambda$ for each $P_i \in \mathcal{P}_{(1)} \setminus \mathcal{P}_{(1)}^{\mathcal{A}}$.
Honest parties:

1. Sample random sharings $\llbracket x_k \rrbracket, \llbracket y_k \rrbracket, \llbracket z_k \rrbracket$, with $x_k, y_k, z_k \leftarrow \{0,1\}$ such that $z_k = x_k \cdot y_k, k \in [m]$.

Corrupt parties: Corrupt parties choose their own randomness in the MAC shares.

Fig. 9. Functionality for triples generation.

7.1 Half Authenticated Triples

Here we show how $\Pi^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$ securely computes cross terms in triples. The main difficulty arises from modelling the leakage due to using short keys in the real world, and proving that it cannot be distinguished from uniformly random. Looking at individual parties, security relies on the fact that on step 6a of the protocol $s^{i,j}_k$ is a fresh, random sharing of zero and hence $y^{i,j}_k$ is perfectly masked. Nevertheless, when considering the *joint* leakage from all honest parties, the $\mathcal{L}$-DRSD assumption kicks in and requires a more thoughtful consideration.

Functionality for Half Authenticated Triples - $\mathcal{F}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$

PARAMETERS: r, number of multiplications; ℓ, key length. The functionality runs between a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, containing two (possibly overlapping) subsets $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$, such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party and an adversary $\mathcal{A}$.

1. The functionality receives correlations Δ^i from each $P_i \in \mathcal{P}_{(h)}$, picks random $[x_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}_{\Delta}$ for $k \in [r]$ and sends the relevant part of the representation to the relevant parties.
2. The functionality receives bits $\{y^{i,j}_k\}_{j\in(\mathcal{P}_{(1)}\setminus\{P_i\}),k\in[r]}$ from each $P_i \in \mathcal{P}_{(h)}$. Then it samples random bits $\{v^{\tau}_k\}_{\tau\in(\mathcal{P}_{(h)}\cup\mathcal{P}_{(1)}),k\in[r]}$, such that:

$$\sum_{\tau\in(\mathcal{P}_{(h)}\cup\mathcal{P}_{(1)})} v^{\tau}_k = \sum_{i\in\mathcal{P}_{(h)}} \sum_{j\in(\mathcal{P}_{(1)}\setminus\{P_i\})} x^j_k \cdot y^{i,j}_k,$$

   and sends $v^{\tau}_k, k \in [r]$, to $P_\tau \in (\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)})$.

Global Key Queries: Upon receiving (i, P) from $\mathcal{A}$, where P is an efficiently computable predicate $P : \{0,1\}^\ell \to \{0,1\}$, output 1 to $\mathcal{A}$ if $P(\Delta^i) = 1$. Otherwise, output 0 to $\mathcal{A}$ and abort.

Fig. 10. Functionality for Half Authenticated Triples

Security is showed in the following theorem, proved in the the full version.

Theorem 7.1. *Protocol $\Pi^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$ securely implements $\mathcal{F}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$ in the $(\mathcal{F}_{\mathsf{aBit}}, \mathcal{F}_{\mathsf{Zero}})$-hybrid model as long as $\mathcal{L}$-$\mathsf{DRSD}_{r,h,\ell}$ is secure.*

7.2 Correct Non-Leaky Authenticated Triples

Here we describe the protocol Π_{Triple} (Figure 12) to create m correct random authenticated triples with compact MACs $[\![x_k]\!], [\![y_k]\!], [\![z_k]\!], k \in [m]$.

First, parties in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ call $\mathcal{F}_{\mathsf{aBit}}$ obtaining $m' = m \cdot B^2 + c$ random authenticated bits $\{[y_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}\}_{k\in m'}$, where B and c are parameters of the sub-protocol

Protocol $\Pi^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$

REQUIRE: r, number of multiplications; ℓ, key length. The protocol runs between a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, containing two (possibly overlapping) subsets $\mathcal{P}_{(h)}, \mathcal{P}_{(1)}$, such that $\mathcal{P}_{(h)}$ has at least h honest parties and $\mathcal{P}_{(1)}$ has at least one honest party.
INPUT: From all $P_i \in \mathcal{P}_{(h)}$: $\Delta^i \in \mathbb{F}_2^\ell$ and $\boldsymbol{y}^{i,j} = (y_1^{i,j}, \ldots, y_r^{i,j}) \in \mathbb{F}_2^m$, where j ranges for every $P_j \in \mathcal{P}_{(1)} \setminus \{P_i\}$.
COMMON INPUT: Random functions $\mathsf{H}_{i,j} : [r] \times \{0,1\} \times \{0,1\}^\ell \to \{0,1\}$ for $P_i \in \mathcal{P}_{(h)}, P_j \in \mathcal{P}_{(1)}$.
OUTPUT: Values $[x_1]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, \ldots, [x_r]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ and shares $v_1^j, \ldots, v_r^j$ for $P_j \in \mathcal{P}_{(1)}$.

1. Parties call $\mathcal{F}_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m,\ell}$ on input random $\{x_k^j\}_{k\in[r]}$ from $P_j \in \mathcal{P}_{(1)}$ to obtain $[x_k]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, k \in [r]$.
2. Parties initialize a new $\mathcal{F}_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}$ instance with $\Delta^i + \tilde{\Delta}^i$, and then call this on input the shares $(x_1^j, \ldots, x_r^j)$ to obtain $[x_1]_{\Delta+\tilde{\Delta}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, \ldots, [x_r]_{\Delta+\tilde{\Delta}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$.
3. Each $P_i \in \mathcal{P}_{(h)}$ sets $\boldsymbol{k}_{\tilde{\Delta}}^{i,j}[x_k] = \boldsymbol{k}_\Delta^{i,j}[x_k] + \boldsymbol{k}_{\Delta+\tilde{\Delta}}^{i,j}[x_k]$ for $k \in [r], j \in \{\mathcal{P}_{(1)} \setminus P_i\}$.
4. Each $P_j \in \mathcal{P}_{(1)}$ sets $\boldsymbol{m}_{\tilde{\Delta}}^{j,i}[x_k] = \boldsymbol{m}_\Delta^{j,i}[x_k] + \boldsymbol{m}_{\Delta+\tilde{\Delta}}^{j,i}[x_k]$ for $k \in [r], i \in \{\mathcal{P}_{(h)} \setminus P_j\}$.
5. Parties $P_i \in \mathcal{P}_{(h)}$ call $\mathcal{F}_{\mathsf{Zero}}$ so that each P_i obtains shares $\{(s_1^{i,j}, \ldots, s_r^{i,j})\}_{j\in\mathcal{P}_{(1)}}$ s.t. $\sum_{i\in\mathcal{P}_{(h)}} s_k^{i,j} = 0, \forall k \in [r]$.
6. For each $k \in [r]$, $P_i \in \mathcal{P}_{(h)}$ and $P_j \in \mathcal{P}_{(1)}$:
   (a) $P_i \in \mathcal{P}_{(h)}$ computes:
   $$d_k^{i,j} = \mathsf{H}_{i,j}(k, 0, \boldsymbol{k}_{\tilde{\Delta}}^{i,j}[x_k]) + \mathsf{H}_{i,j}(k, 1, \boldsymbol{k}_{\tilde{\Delta}}^{i,j}[x_k] + \tilde{\Delta}^i) + y_k^{i,j} + s_k^{i,j},$$
   and privately sends it to P_j, for each $P_j \in (\mathcal{P}_{(1)} \setminus \{P_i\})$.
   (b) Each $P_j \in \mathcal{P}_{(1)}$ computes, for $i \in (\mathcal{P}_{(h)} \setminus \{P_j\})$:
   $$t_k^{j,i} = \mathsf{H}_{i,j}(k, x_k^j, \boldsymbol{m}_{\tilde{\Delta}}^{j,i}[x_k]) + x_k^j \cdot d_k^{i,j}.$$
   (c) Each $P_i \in \mathcal{P}_{(h)}$ computes (where $x_k^i = s_k^{i,i} = 0$ if $P_i \notin \mathcal{P}_{(1)}$):
   $$t_k^{i,i} = \sum_{j\in(\mathcal{P}_{(1)}\setminus\{P_i\})} \mathsf{H}_{i,j}(k, 0, \boldsymbol{k}_{\tilde{\Delta}}^{i,j}[x_k]) + x_k^i \cdot s_k^{i,i}.$$
7. Parties in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ call $\mathcal{F}_{\mathsf{Zero}}$ so that each P_τ obtains shares $(\rho_1^\tau, \ldots, \rho_r^\tau)$ such that $\sum_{\tau\in(\mathcal{P}_{(h)}\cup\mathcal{P}_{(1)})} \rho_k^\tau = 0, \forall k \in [r]$.
8. For $k \in [r]$, each $P_j \in \mathcal{P}_{(1)}$ computes $v_k^j = \sum_{i\in\mathcal{P}_{(h)}} t_k^{j,i} + \rho_k^j$ and each $P_i \in (\mathcal{P}_{(h)} \setminus \mathcal{P}_{(1)})$ sets $v_k^i = t_k^{i,i} + \rho_k^i$.

Fig. 11. Protocol for Half Authenticated Triples

$\Pi_{\mathsf{TripleBucketing}}$ (Figure 13). Then, each $P_j \in \mathcal{P}_{(1)}$ reshares their values y_k^j to parties in $\mathcal{P}_{(h)}$ obtaining $[\hat{y}_k]_{k\in m'}^{\mathcal{P}_{(h)},\mathcal{P}_{(h)}}$ such that $\sum_{i\in\mathcal{P}_{(h)}} \hat{y}_k^i = y_k, k \in [m]$.
This allows $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ to call $\mathcal{F}_{\mathsf{HalfAuthTriple}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}$ $\hat{m} = m/r$ times, on inputs $\{\hat{y}_{(\iota-1)\cdot r+k}\}_{k\in[r]}$, for each $\iota \in \hat{m}$. The outputs of each of these calls are the sharings $v_{(\iota-1)\cdot r+k}^\tau$, $\tau \in$

$\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ and $k \in [r]$, of r cross terms products, i.e.

$$\sum_{\tau \in \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}} v^{\tau}_{(\iota-1)\cdot r+k} = \sum_{i \in \mathcal{P}_{(h)}} \sum_{j \in \mathcal{P}_{(1)}} x^{j}_{(\iota-1)\cdot r+k} \cdot \hat{y}^{i}_{(\iota-1)\cdot r+k}.$$

Notice that the number r of cross terms computed by $\mathcal{F}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$ depends on the leaky DRSD problem, and for this reason the protocol needs to call the functionality $\hat{m}$ times to obtain all the m' outputs it needs.

After this, parties in $\mathcal{P}_{(h)}$ reshare all the $v^{i}_{k}, k \in m'$ to $\mathcal{P}_{(1)}$, so that each $P_j \in \mathcal{P}_{(1)}$ gets $\hat{v}^{j}_{k}, k \in [m']$, where

$$\sum_{j \in \mathcal{P}_{(1)}} \hat{v}^{j}_{k} = \sum_{j \in \mathcal{P}_{(1)}} x^{j}_{k} \sum_{i \in \mathcal{P}_{(1)} \setminus j} y^{i}_{k} = \sum_{\tau \in \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}} v^{\tau}_{k}, \tag{1}$$

so that parties in $\mathcal{P}_{(1)}$ can locally add shares $x^{j}_{k} \cdot y^{j}_{k}$ to $\hat{v}^{j}_{k}$ obtaining z^{j}_{k}, $k \in [m']$.

Finally, $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ call $\mathcal{F}_{\mathsf{aBit}}$ to obtain $[z_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$, and run the $\Pi_{\mathsf{TripleBucketing}}$ subprotocol. This subprotocol is similar to the bucket-based cut-and-choose technique introduced by Larraia et al. [LOS14] and optmized by Frederiksen et al. [FKOS15], but adapted to run with two committees. It takes as input $m' = B^2 \cdot m + c$ triples. First, in Step I and II, it ensures that all the triples are correctly generated sacrificing $B \cdot m \cdot (B-1) + c$ triples, and then (Step III) it uses random bucketing technique to remove potential leakage on the x_k values obtaining m private and correct triples. All the MACs on previously opened values are eventually checked (Step IV) calling the **Batch Check** commmand in $\Pi_{[\mathsf{Open}]}$ (Figure 1). Finally, on that last step, the remaining triples are converted to SPDZ-style triples in $\mathcal{P}_{(1)}$ using $\Pi_{\mathsf{MACCompact}}$.

Correctness easily follows form the discussion above:

$$\sum_{j \in \mathcal{P}_{(1)}} z^{j}_{k} = \sum_{j \in \mathcal{P}_{(1)}} x^{j}_{k} \cdot y^{j}_{k} + \hat{v}^{j}_{k}, \tag{2}$$

where $\hat{v}^{j}_{k}$ is the re-sharing inside $\mathcal{P}_{(1)}$ of $\mathcal{F}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}}$'s output. More precisely, using Equation 1 we can rewrite Equation 2 as follows:

$$\begin{aligned}\sum_{j \in \mathcal{P}_{(1)}} z^{j}_{k} &= \sum_{j \in \mathcal{P}_{(1)}} x^{j}_{k} \cdot y^{j}_{k} + \sum_{j \in \mathcal{P}_{(1)}} x^{j}_{k} \cdot \sum_{i \in \mathcal{P}_{(1)} \setminus j} y^{i}_{k} \\ &= \sum_{j \in \mathcal{P}_{(1)}} x^{j}_{k} \cdot \big(y^{j}_{k} + \sum_{i \in \mathcal{P}_{(1)} \setminus j} y^{i}_{k}\big) = \big(\sum_{j \in \mathcal{P}_{(1)}} x^{j}_{k}\big) \cdot \big(\sum_{j \in \mathcal{P}_{(1)}} y^{j}_{k}\big).\end{aligned}$$

Security is showed in the following theorem, proved in the full version.

Theorem 7.2. *Protocol Π_{Triple} securely implements $\mathcal{F}^{m,\ell}_{\mathsf{Triple}}$ in the $(\mathcal{F}_{\mathsf{Rand}}, \mathcal{F}_{\mathsf{aBit}}, \mathcal{F}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}_{\mathsf{HalfAuthTriple}})$-hybrid model.*

Parameters: Based on the analysis from previous works [FKOS15, FLNW17, WRK17a], we choose $B = 3$ and 4, to guarantee security except with probability 2^{-64} in our estimations. The additional cut-and-choose parameter c can be as low as 3, so is insignificant as we initially need $m' = B^2 m + c$ triples to produce m final triples.

Protocol for triples generation - $\Pi_{\mathsf{Triple}}^{m,r}$

PARAMETERS: Let B, c be parameters as needed for $\Pi_{\mathsf{TripleBucketing}}$ and $m' = m \cdot B^2 + c$. Let r be as needed for the security of the leaky DRSD problem in $\mathcal{F}_{\mathsf{HalfAuthTriple}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}$.
Initialize: Parties initialize $\mathcal{F}_{\mathsf{aBit}}$ which outputs Δ^i to each $P_i \in \mathcal{P}_{(h)}$.
Triple Computation:

1. Parties in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ call $\mathcal{F}_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m',\ell}$ on input random $\{y_k^j\}_{k\in[m']}$ from $P_j \in \mathcal{P}_{(1)}$ and obtains random authenticated shares $\{[y_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}\}_{k\in[m']}$.
2. Reshare $y_k, k \in [m']$ from $\mathcal{P}_{(1)}$ to $\mathcal{P}_{(h)}$ as follows:
   - Each $P_j \in \mathcal{P}_{(1)} \setminus \mathcal{P}_{(h)}$ secret shares $y_k^j = \sum_{i\in\mathcal{P}_{(h)}} y_k^{i,j}$ and sends $y_k^{i,j}$ to $P_i \in \mathcal{P}_{(h)}$.
   - Each $P_i \in \mathcal{P}_{(h)}$ sets $\hat{y}_k^i = y_k^i + \sum_{j\in\mathcal{P}_{(1)}} y_k^{i,j}$, where $y_k^i = 0$ if $P_i \notin \mathcal{P}_{(1)}$.
3. Let $\hat{m} = \lceil m'/r \rceil$. For $\iota \in [\hat{m}]$, the parties call $\mathcal{F}_{\mathsf{HalfAuthTriple}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},r,\ell}$ on input $\{\hat{y}_{(\iota-1)\cdot r+k}^{i,j}\}_{k\in[r],j\in\mathcal{P}_{(1)}}$ from $P_i \in \mathcal{P}_{(h)}$, where $\hat{y}_{(\iota-1)\cdot r+k}^{i,j} = \hat{y}_{(\iota-1)\cdot r+k}^{i}, \forall j \in \mathcal{P}_{(1)}$, to obtain random $\{[x_{(\iota-1)\cdot r+k}]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}\}_{k\in[r]}$, and $\{v_{(\iota-1)\cdot r+k}^{\tau}\}_{\tau\in\mathcal{P}_{(h)}\cup\mathcal{P}_{(1)},k\in[r]}$, $\forall \iota \in [\hat{m}]$.
4. Reshare $v_k^\tau, k \in [m']$, from $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ to $\mathcal{P}_{(1)}$ as follows:
   - Each $P_i \in \mathcal{P}_{(h)} \setminus \mathcal{P}_{(1)}$ secret shares $v_k^i = \sum_{j\in\mathcal{P}_{(1)}} v_k^{j,i}$ and sends $v_k^{j,i}$ to $P_j \in \mathcal{P}_{(1)}$.
   - Each $P_j \in \mathcal{P}_{(1)}$ sets $\hat{v}_k^j = v_k^j + \sum_{i\in\mathcal{P}_{(h)}\setminus\mathcal{P}_{(1)}} v_k^{j,i}$.
5. For $k \in [m']$ each $P_j \in \mathcal{P}_{(1)}$ computes $z_k^j = x_k^j \cdot y_k^j \oplus \hat{v}_k^j$, where $y_k^j = 0$ if $P_j \notin \mathcal{P}_{(h)}$. Parties in $\mathcal{P}_{(h)} \cup \mathcal{P}_{(1)}$ call $\mathcal{F}_{\mathsf{aBit}}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)},m',\ell}$ on input $\{z_k^j\}_{k\in[m']}$ from $P_j \in \mathcal{P}_{(1)}$ to obtain $\{[z_k]^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}\}_{k\in[m']}$.

Triple Checking: Run $\Pi_{\mathsf{TripleBucketing}}$ to output m correct and secure triples.

Fig. 12. Protocol for Triples

Subprotocol $\Pi_{\mathsf{TripleBucketing}}$

The protocol takes as input $m' = B^2m + c$ triples, which may be incorrect and/or have leakage on the x component, and produces m triples which are guaranteed to be correct and leakage-free.
B determines the bucket size, whilst c determines the amount of cut-and-choose to be performed.
Input: Start with the shared triples $\{[x_k]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, [y_k]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, [z_k]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}\}_{k\in[m']}$.
Output: Correct, leakage-free and SPDZ-style triples $\{[\![x_k]\!], [\![y_k]\!], [\![z_k]\!]\}_{k\in[m]}$.
I: Cut-and-choose: Using $\mathcal{F}_{\mathsf{Rand}}$, the parties select at random and open c triples using $\Pi_{[\mathsf{Open}]}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ (Figure 1). If any triple is incorrect (i.e. if $x \cdot y \neq z$), abort.

II: Check correctness: The parties now have B^2m unopened triples.

1. Use $\mathcal{F}_{\mathsf{Rand}}$ to sample a random permutation on $\{1, \ldots, B^2m\}$, and randomly assign the triples into mB buckets of size B, accordingly.
2. For each bucket, check correctness of the first triple in the bucket, say $T = ([x], [y], [z])$, by performing a pairwise sacrifice between T and every other triple in the bucket. Concretely, to check correctness of T by sacrificing $T' = ([x'], [y'], [z'])$:
   (a) Compute $[d] = [x] + [x']$ and $[e] = [y] + [y']$ and call $(\mathsf{Open}, [d], \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)})$, $(\mathsf{Open}, [e], \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)})$ in $\Pi_{[\mathsf{Open}]}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$.
   (b) Using the opened values, compute $[f] = [z] + [z'] + d \cdot [y] + e \cdot [x] + d \cdot e$.
   (c) Call $(\mathsf{Open}, [f], \mathcal{P}_{(1)})$ in $\Pi_{[\mathsf{Open}]}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ and check that $f = 0$. Otherwise, the parties abort.

III: Remove leakage: Taking the first triple in each bucket from the previous step, the parties are left with Bm triples. They remove any potential leakage on the $[x]$ bits of these as follows:

1. Place the triples into m buckets of size B.
2. For each bucket, combine all B triples into a single triple. Specifically, combine the first triple $([x], [y], [z])$ with $T' = ([x'], [y'], [z'])$, for every other triple T' in the bucket:
   (a) Compute $[d] = [y] + [y']$ and call $(\mathsf{Open}, [d], \mathcal{P}_{(h)} \cup \mathcal{P}_{(1)})$ in $\Pi_{[\mathsf{Open}]}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$.
   (b) Compute $[z''] = d \cdot [x'] + [z] + [z']$ and $[x''] = [x] + [x']$.
   (c) Output the triple $[x'']_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, [y]_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}, [z'']_\Delta^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$.

IV: Check MACs and compact them: Call **Batch Check** in $\Pi_{[\mathsf{Open}]}^{\mathcal{P}_{(h)},\mathcal{P}_{(1)}}$ for every item that was opened in Steps I-III. If the batched MAC check fails, abort. Otherwise call $\Pi_{\mathsf{MACCompact}}$ on input the first triple from each of the m buckets in the previous stage, which are renamed and output as $\{[\![x_k]\!], [\![y_k]\!], [\![z_k]\!]\}_{k\in[m]}$.

Fig. 13. Checking correctness and removing leakage from triples with cut-and-choose

8 Complexity Analysis

We now analyse the complexity of our protocol and compare it with the state-of-the-art actively secure MPC protocols with dishonest majority. As our online phase is essentially the same (even better) than that of SPDZ and TinyOT mixed with committees, we focus on the preprocessing phase.

Furthermore, since the underlying computational primitives in our protocol are very simple, the communication cost in the triple generation algorithm will be the overall bottleneck. We compare the communication cost of our triple generation algorithm with that of the corresponding multiparty Tiny-OT protocol by Wang et al. [WRK17b].

The main cost for producing m triples in this work, is $3mB^2$ calls to $\mathcal{F}_{\mathsf{aBit}}$ using keys $\Delta^i \in \{0,1\}^\ell$, plus mB^2 calls to $\mathcal{F}_{\mathsf{aBit}}$ using new keys $\Delta^i + \tilde{\Delta}^i \in \{0,1\}^\ell$ every r triples. The latter calls under new keys are more expensive, as the setup costs that incurs is roughly $128 \cdot \ell \cdot |\mathcal{P}_{(h)}| \cdot |\mathcal{P}_{(1)}|$ bits and is amortized only across those r triples. Measuring the cost of $\mathcal{F}_{\mathsf{aBit}}$ after setup as $|\mathcal{P}_{(h)}| \cdot |\mathcal{P}_{(1)}| \cdot \ell$ bits, we obtain an amortized communication complexity of $B^2 \cdot |\mathcal{P}_{(h)}| \cdot |\mathcal{P}_{(1)}| \cdot \ell \cdot (3 + (r + 128)/r)$ bits per triple.

The main cost for producing m triples in [WRK17b] is $3mB$ calls to their long-key equivalent of $\mathcal{F}_{\mathsf{aBit}}$ with long keys, plus sending $2mB$ outputs of a hash function. On the other hand, all their communication is within the smaller committee $\mathcal{P}_{(1)}$. Their main (amortized) cost is then of $B \cdot |\mathcal{P}_{(1)}|^2 \cdot 128 \cdot (3 + 2)$ bits per triple. Define $\alpha = |\mathcal{P}_{(h)}|/|\mathcal{P}_{(1)}|$. We can then conclude that the improvement in communication complexity of our work w.r.t. WRK is roughly that of a multiplicative factor of:

$$\frac{128 \cdot 5}{\alpha \cdot B \cdot \ell \cdot (4 + 128/r)}$$

Given the total number of parties n and honest parties h, we first consider the case of two deterministic committees $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$ such that $|\mathcal{P}_{(h)}| = n$ and $|\mathcal{P}_{(1)}| = n-h+1$, respectively. To give a fair comparison, we have chosen the parameters in such a way that $n - h + 1$ in our protocol is equal to n in WRK. The estimated amortized costs in kbit of producing triples are given in Table 1. Notice that given n and h, the key lenght ℓ and the number of triples r are established according to the corresponding leaky-DRSD instance with κ bits of security. We consider $\kappa = 128$ and bucket size $B = 3$ and 4.
As we can see from the table, the improvement of our protocol over WRK becomes greater as (n, h) increase (and ℓ consequentely decreases). The key lenght greatly influences the communication cost as a smaller ℓ reduces significantly the cost of computing the pairwise OTs needed both for triple generation and authentication.

When n is larger we can use random committees $\mathcal{P}_{(h)}$ and $\mathcal{P}_{(1)}$ such that, except with negligible probability $2^{-\lambda}$, $\mathcal{P}_{(h)}$ has at least $h_2 \leq h$ honest parties and $\mathcal{P}_{(1)}$ has at least 1 honest party. Let $|\mathcal{P}_{(h)}| = n_2$, $|\mathcal{P}_{(1)}| = n_1$ and $\lambda = 64$, Table 2 compares the communication cost of our triple generation protocol with random committees with WRK, where we take $n = n_1$.

Varying the size of the committee $\mathcal{P}_{(h)}$, and the number h_2 of honest parties within $\mathcal{P}_{(h)}$, we obtain a tradeoff: with a larger committee we obtain a larger committee size n_2 and lower overall communication complexity, but on the other hand there are more parties interacting, which may introduce bottlenecks in the networking. Figure 14 illustrates this with 500 and 1000 parties in total and 350 and 850, respectively, corruptions.

# parties n (honest)	30 (12)	50 (20)	70 (20)	100 (30)	150 (30)	200 (40)
ℓ	16	16	16	8	$(8, 300)$	$(7, 400)$
WRK $B = 3$	656	1785	4896	9542	27878	49959
WRK $B = 4$	876	2381	6528	12723	37171	65946
Ours $B = 3$	**381**	**950**	**2188**	**2481**	**6342**	**9413**
Ours $B = 4$	**677**	**1689**	**3890**	**4411**	**11275**	**16733**

Table 1. Amortized communication cost (in kbit) of producing triples in our protocol and WRK.

$(n, h, \lvert\mathcal{P}_{(1)}\rvert)$	$(h_2, \ell) = (20, 11)$		$(h_2, \ell) = (50, 6)$		$(h_2, \ell) = (80, 1)$		$(h_2, \ell) = (110, 1)$		$(h_2, \ell) = (150, 1)$		
	n_2	Ours	n_2	Ours	n_2	Ours	n_2	Ours	n_2	Ours	WRK
$(300, 100, 89)$	167	**7141**	240	**5270**	280	**4486**					15037
$(500, 150, 108)$	211	**10950**	316	**8420**	396	**7698**	456	**4240**			50700
$(800, 200, 139)$	275	**18366**	417	**14301**	533	**13335**	630	**7539**	733	**6397**	36829
$(1000, 200, 179)$	351	**30188**	531	**23451**	675	**21748**	796	**12266**	922	**10363**	61175

Table 2. Amortized costs in kbit for triple generation with n parties and h honest parties using two random committees of sizes n_1, n_2 with 1 and h_2 honest parties.

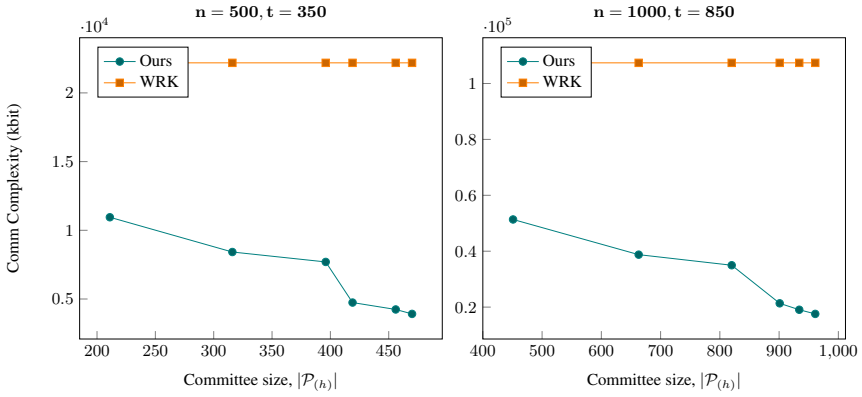

Fig. 14. Varying the larger committee size with total number of parties and corruptions $(n, t) = (500, 350)$ and $(1000, 850)$.

References

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

BCD+09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation

goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Heidelberg, February 2009.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 169–188. Springer, Heidelberg, May 2011.

BLN+15. 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. Cryptology ePrint Archive, Report 2015/472, 2015. http://eprint.iacr.org/2015/472.

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

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 213–230. Springer, Heidelberg, March 2008.

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

DKL+13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Heidelberg, September 2013.

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

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

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, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 711–735. Springer, Heidelberg, November / December 2015.

FLNW17. Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 225–255. Springer, Heidelberg, April / May 2017.

HOSS18. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Tinykeys: A new approach to efficient multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 3–33. Springer, Aug. 2018.

HSS17. 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, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 598–628. Springer, Heidelberg, December 2017.

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

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

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 830–842. ACM Press, October 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, pages 158–189, 2018.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.

KSS12. Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX*, pages 285–300, 2012.

LOS14. Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multiparty computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, Heidelberg, August 2014.

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

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, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2012.

NST17. Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In *NDSS 2017*. The Internet Society, February / March 2017.

WMK17. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster secure two-party computation in the single-execution setting. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 399–424. Springer, Heidelberg, April / May 2017.

WRK17a. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 21–37. ACM Press, October / November 2017.

WRK17b. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 39–56. ACM Press, October / November 2017.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.