Beaver Triple Preprocessing For Efficient Decentralized Matching

Evelyn Lianto, Vasawat Rawangwong

May 2026

1 Introduction

Secure Multiparty Computation (MPC) enables multiple parties to jointly compute a function over their private inputs without revealing those inputs to one another. However, many MPC protocols are considered impractical in real-world deployments due to the high communication cost of non-linear operations. In many secret-sharing-based MPC schemes, addition and scalar multiplication can be performed locally on shares without interaction between parties. In contrast, multiplication and comparison operations typically require communication rounds between participants, making them significantly more expensive. Since many complex functions require a large number of multiplication and comparison gates, communication overhead often becomes the primary bottleneck in MPC systems, particularly in malicious or decentralized settings where additional verification mechanisms are required.

To reduce online communication costs, many modern MPC protocols adopt a preprocessing paradigm that separates computation into a data-independent offline phase and a latency-sensitive online phase. One important example is Beaver multiplication, which uses preprocessed correlated randomness known as *Beaver triples* [1]. Given authenticated shares of a Beaver triple and secret shares of two values, parties can evaluate multiplication using only a constant number of online interaction rounds. This significantly reduces online latency compared to traditional MPC multiplication protocols such as BGW multiplication. However, generating Beaver

triples securely without relying on a trusted dealer remains a major challenge in decentralized environments.

The generation of Beaver triples may rely on a "Trusted Dealer" to generate random shares of Beaver triples. However, the most practical applications of MPC arise precisely in settings where no trusted central authority exists. As such, this "Trusted Dealer" often does not exist in many scenarios. Electronic voting systems are often not trusted by the public due to their lack of transparency and dependence on the system's administrators, making voters believe that ballots can be easily modified. Moderators can often tamper with the results of competitions or games if they are biased towards certain teams. Auctions can be rigged as winners are difficult to verify as little information about bid amounts are revealed to the public for privacy reasons. As a result, efficient decentralized protocols for generating authenticated Beaver triples are essential for enabling practical low-latency MPC in untrusted environments

In this report, we study the use of Beaver Triple Preprocessing MPC for decentralized low-latency online matching systems. We consider a secure investor-fund matching scenario in which investors privately submit preference vectors while funds privately submit feature embeddings. The compatibility score between an investor and a fund is computed as the dot product of their respective vectors, followed by a secure row-wise argmax operation to determine the best matches. This workload is particularly suitable for evaluating preprocessing-based MPC because secure comparisons require many multiplication gates and therefore incur substantial communication costs in traditional MPC protocols. We implement and benchmark a preprocessing-based protocol using MASCOT-generated Beaver triples and compare its online latency against malicious Shamir-based MPC using BGW-style multiplication.

2 Related Work

2.1 Secret Sharing

Secret sharing refers to the cryptographic method of distributing parts of a secret, called shares, among a group in such a way that any sufficiently large subset of distributed shares are sufficient to reconstruct the secret, while subsets which do not reach this threshold reveal no information about the

secret at all. A particular scheme is called a t-out-of-n secret sharing scheme if the secret can be reconstructed from any subset of at least t shares, while no information about the secret is revealed from knowing any subset of less than t shares. The case $t = 1$ is trivial, as the only possibility is to share the secret to every party. For $t = n$, many schemes have been devised to ensure both correctness of reconstruction and security. For example, we could generate random bit strings $s_1, \ldots, s_{n-1}$ of length equal to n. Then, party i receives the share s_i, where $s_n = s \oplus s_1 \oplus \cdots \oplus s_{n-1}$.

For $1 < t < n$, the scheme must be carefully designed for security, while maintaining correctness even when not all shares are used. One such scheme is Shamir's Secret Sharing Scheme [7]. This scheme constructs a random polynomial f with degree $t-1$ and coefficients in $GF[p]$ such that $f(0) = s$. Namely, the polynomial can be written in the form

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + s$$

Then for $i = 1, \ldots, n$, the share for party i is the evaluation of the polynomial at i, which is $f(i)$. Then the polynomial f (or just the value $f(0)$, as the entire polynomial is not required for this task) can be reconstructed by interpolation, given at least t points where the polynomial is evaluated at. Formally, we have: $s = f(0) = \sum_{j=1}^{t-1} \prod_{k \neq j} \frac{f(i_k)}{f(i_k) - f(i_j)}$ A nice property of polynomial interpolation is that given the equations of evaluating the polynomial at t different points, we get an independent linear system of equations with t variables corresponding to the coefficients of the polynomial. Therefore, it is impossible to interpolate the polynomial when given only $t-1$ points. More specifically, if the coefficients of the non-constant terms of the polynomial were chosen uniformly, then each possible value of $f(0)$ is equally likely. Thus, the security condition of the scheme is satisfied.

However, secret sharing schemes such as Shamir's scheme would fail if some parties are compromised. Malicious parties often have the option to submit false shares, which if left unverified, could cause an incorrect result due to interpolating polynomials on false input/output pairs. These errors are often left unnoticed until evaluations are done on the secret, which often happens much later in the process. This makes it hard to both recompute the correct result and determine who the malicious party was. Many schemes have been devised to immediately verify the correctness of shares at the moment when shares are submitted. Such schemes are called Verified Secret Sharing Schemes, which often assigns a commitment hash to each party's

share, allowing them to be verified at will. Feldman's scheme [4], uses the discrete-log hardness assumption to assign commitments corresponding to the coefficients of the polynomial f of Shamir's scheme. g is chosen to be a generator of a cyclic group G with prime order. The commitments are

$$\begin{aligned} c_0 &= g^s \\ c_1 &= g^{a_1} \\ &\vdots \\ c_{t-1} &= g^{a_{t-1}} \end{aligned}$$

And so each share s_i can be verified as follows:

$$g^{s_i} = c_0 c_1^i c_2^{i^2} \dots c_{t-1}^{i^{t-1}} = g^{\sum_{j=0}^{t-1} a_j i^j} = g^{f(i)}$$

Therefore, this scheme is secure under computationally bounded malicious adversaries. For computationally unbounded adversaries, the information $c_0 = g^s$ leaks information about s. Other Verified Secret Sharing schemes exist which do not leak information about the secret at all, such as Pederson's scheme [6].

2.2 Multiplication in MPC Protocols

The natural question which arises from secret sharing protocols is whether we could perform operations on the secret in such a way that no party learns about the secret, but their shares represent the result of applying the operation to the secret itself. This is called Secure Multiparty Computation (MPC). In most secret sharing schemes, we find that certain operations are easy to implement than others, provided that parties are honest and follow protocol. For example, using Shamir's Secret Sharing scheme, shares for addition and scalar multiplication of secrets are able to be computed locally. Each party who holds shares $f(i)$ and $g(i)$ of the secrets $f(0)$ and $g(0)$ can receive their share of the secrets $f(0) + g(0)$ and $cf(0)$ for some constant c by computing $f(i) + g(i)$ and $cf(i)$ respectively. None of these operations require the party to communicate with other parties, which results in great efficiency.

However, the multiplication in MPC does not have a straightforward implementation like addition and scalar multiplication. When using Shamir's Secret Sharing scheme, we find that the polynomial $f(x)g(x)$ has degree $2t-2$,

so at least $2t-1$ shares of the form $f(i)g(i)$ are required to reconstruct the product of secrets, which breaks correctness. Certain protocols, such as the Ben-Or, Goldwasser and Wigderson (BGW) multiplication protocol [2], has at least $2t-1$ parties reshare the product of their share $f(i)g(i)$ using the same t-out-of-n Shamir's Secret Sharing scheme to every other party. This ensures that knowing the reshares for t groups reveals the shares $f(i)g(i)$ for $2t-1$ groups, which is therefore sufficient to reconstruct the secret. However, we find that the number of distinct messages that must be communicated is $\Theta(tn)$, which could be up to $\Theta(n^2)$ if $t = \Theta(n)$, (which is a natural choice for brute force attacks). This communication cost adds up the more multiplication gates the function undergoing MPC requires, which often makes the BGW multiplication protocol impractical in real-world scenarios.

3 The Preprocessing Paradigm in Decentralized MPC

3.1 Beaver Triples

Beaver triples are a preprocessing primitive used to accelerate secure multiplication in MPC protocols. A Beaver triple [1] consists of secret shares of three correlated random values (A, B, C) such that C = AB. Suppose parties hold secret shares of two values x and y together with shares of a Beaver triple. Multiplication can then be evaluated using the following protocol

1. Each party locally compute and reveals the masked values $x - A$ and $y - B$.

2. Each party then locally computes a share of the product using $xy = (x-A)(y-B) + (x-A)B + (y-B)A + AB$

The key advantage of Beaver multiplication is that it significantly reduces online interaction complexity. During online execution, parties only need to reveal the masked differences $x - A$ and $y - B$, after which the remaining computation can be performed locally. This requires only a constant number of online communication rounds per multiplication gate. If each revealed value is broadcast to all n parties, the online communication consists of $\Theta(n)$ total transmitted messages per multiplication rather than the $\Theta(n^2)$ communication required by BGW-style resharing protocols. Consequently,

preprocessing-based MPC substantially reduces online latency even though the expensive communication is merely shifted into the offline phase rather than eliminated entirely.

Correctness of this algorithm follows from the correctness of the shares. However, privacy is guaranteed if and only if the Beaver Triple shares are generated uniformly. (If A_i and B_i were not truly uniform, then revealing $x_i - A_i$ and $y_i - B_i$ leaks information about x_i and y_i.) Additionally, each Beaver triple must be consumed after being used in a multiplication operation to prevent leakage of information. Therefore, we could shift the costs to the offline phase of the algorithm by preprocessing sufficiently many Beaver Triples.

3.2 MASCOT Protocol

A couple of issues arise when trying to generate Beaver triples without the help of a trusted central authority. The first is the verification concern: that malicious parties may submit false shares, which can lead to the incorrect reconstruction of $x - A$ and $y - B$, causing an error in the result from multiplication. The second is the correctness concern: while it may be easy to generate shares of a uniformly random secret, the three secrets in a beaver triple (A, B, AB) are correlated. It is not obvious how untrusted parties can construct shares of the product without revealing any information about the secret. The third is the security concern: that malicious parties may be able to manipulate the randomness behind the generation of the Beaver triple (A, B, AB), which breaks the security constraint. Fortunately, the MASCOT protocol [5] solves all of these problems.

The MASCOT protocol is a protocol for the generation of the shares of the product of a secret given their shares, even in a malicious setting. We describe how the MASCOT protocol solves the three problems mentioned previously at a high level:

3.2.1 Verification

MASCOT uses the SPDZ protocol [3] to verify the correctness of shares. Each share x_i of the secret x, the protocol attaches a random MAC share m_i and the fixed MAC key share Δ_i to each share such that the MAC relation $m = x \cdot \Delta$ holds. Note that these shares are additive, meaning that $m = \sum_i m_i$ and $\Delta = \sum_i \Delta_i$. Therefore, once the secret x is reconstructed, the

validity of the shares can be verified by checking whether $\sum_i m_i - x \cdot \Delta_i = 0$. This solves the issue of verification.

3.2.2 Correctness

MASCOT uses oblivious product evaluation, a version of oblivious transfer (OT), to create shares of the product. The two party protocol to create shares of the product $a \cdot b$, where one party holds a and the other holds b, is as follows: The two parties perform k sets of oblivious transfer (OT) on k bit strings as follows: The first party (who holds a) samples a random t_i and inputs t_i and $t_i + a$ for each $i = 1, \ldots, k$. The other party (who holds b) inputs the bit decomposition of $(b_1, \ldots, b_k)$ of b and receives $q_i = t + b_i \cdot a$ for each i. Each person then computes the inner product of their values ($(q_i)_i$ and $(-t_i)_i$) with a gadget vector g to get $q + t = ab$. To generate the product share AB for n parties, we can simply run this protocol between every pair of parties. Although this may seem expensive, recall that each party has their own computing resources, so this operation is often parallelizable.

To generate the shares for the MACs, MASCOT uses a more efficient variation of the above protocol called Correlated Oblivious Product Evaluation (COPE). Since the correlation Δ is fixed, the sender can input k pairs of random λ bit seeds, which requires performing the k OTs only once. In each iteration, the sender can use a pseudorandom function (PRF) to expand random seeds to k bits of random OTs and send the masked correlation instead. The receiver then uses this to adjust their own PRF output, which results in k correlated OTs. This allows future secret sharings with Δ to be fast. As a consequence of this protocol, the sharings are also guaranteed to be pseudorandom and secure.

3.2.3 Security

Shares of A, B, Δ can simply be generated by each party independently generating a uniformly random additive share. Provided that at least one party follows this protocol, the secret is guaranteed to be uniformly random. The oblivious product evaluation and COPE algorithms also guarantee randomness and pseudorandomness respectively (although we only use the latter for verification, not the multiplication itself).

Then the general protocol for fast online MPC multiplication using MASCOT can be described as follows:

3.3 Offline Phase

Use the MASCOT protocol to securely generate many shares of Beaver triples (A, B, AB) along with their authentication counterparts $(A \cdot \Delta, B \cdot \Delta, AB \cdot \Delta)$. Each party then stores their shares locally before heading into the online phase.

3.4 Online Phase

When MPC multiplication is required, consume a Beaver triple using Beaver's randomized circuits protocol for MPC multiplication and verify the shares using the attached MACs. For addition and scalar multiplication, do the respective operations locally on both the shares and the MACs.

4 Problem Formulation

4.1 Scenario

We first formalize the scenario. A investor/fund matching event is composed of n fixed investors, each having a private preference vector with dimension d, representing their preferences for each feature of a fund. Each of the m funds also have a private attribute vector of dimension d which is an embedding of their features. Investors and funds are then matched by their compatibility score, which is the dot product of the preference and attribute vectors. The investors do not want to leak their preference vector, as it may reveal their investing strategy. Funds do not want to leak their attribute vector, as it may reveal sensitive internal information. Investors also do not trust each other, and believe that other investors may manipulate the results if they can do so undetected.

4.2 Threat Model

We assume a static malicious adversary that may corrupt a subset of participating investors. Corrupted parties may arbitrarily deviate from the protocol, including submitting malformed shares, performing incorrect local computation, or attempting to infer information about other parties' private inputs from protocol transcripts. We assume that fewer than half of the participating parties are corrupted and that at least one party behaves honestly

during preprocessing, ensuring that jointly generated randomness remains uniformly distributed. We do not consider denial-of-service attacks or selective abort attacks in which parties intentionally terminate the protocol before completion. In particular, we assume that once preprocessing begins, all parties remain online and continue participating throughout the execution of the protocol. We additionally assume that funds submit correctly formed attribute vectors and do not maliciously manipulate their inputs. Our primary focus is therefore on protecting the privacy of investor and fund data while preventing malicious investors from corrupting the matching computation undetected.

5 Protocol

5.1 Offline Phase

The n investors generate secure and verifiable Beaver triples using the MASCOT protocol [5]. Since the number of funds m, the number of features d, and the definition of the function f to be evaluated by MPC is known. The investors are able to know beforehand how many Beaver triples they would require in total. We perform a slight adjustment to the original protocol to generate vectorized shares of Beaver triples instead, which significantly speeds up the online phase.

5.2 Online Phase

For investors with preference vectors $v_1, v_2, \ldots, v_n$ and funds with attribute vectors $u_1, u_2, \ldots, u_n$ (which is secret shared), the function to be computed by MPC is:

$$f(u_1, \ldots, u_n, v_1, \ldots, v_n) = \arg\max \left(\begin{bmatrix} — & u_1^T & — \\ & \vdots & \\ — & u_m^T & — \end{bmatrix} \begin{bmatrix} | & & | \\ v_1 & \ldots & v_n \\ | & & | \end{bmatrix} \right)$$

Where arg max is computed row-wise. Matrix multiplication requires mdn individual multiplications, and arg max requires $m(n-1)$ comparisons, both consuming a lot of beaver triples and computation time as they require many multiplications/bitwise comparisons. To mitigate the issue of runtime, we

evaluate these comparisons using tournament-style comparisons, which allows most of the comparisons to be parallelized. As there are m submitted applicants, we perform this operation m times.

5.3 Complexity Analysis

Under Beaver preprocessing, each multiplication gate requires: $O(1)$ online communication rounds, $\Theta(p)$ communicated values due to broadcasts. Consequently, the online round complexity of the matrix multiplication stage remains O(1) assuming sufficient parallelism, while the total online latency of the matching function is dominated by the O(log n) comparison depth of the argmax stage. During preprocessing, MASCOT requires pairwise oblivious-transfer-based interactions between participating parties. Therefore, the total preprocessing communication grows approximately quadratically in the number of parties: $\Theta(p^2mdn)$ up to protocol-specific constants and batching optimizations.

6 Results

The code for the implementation could be found at `https://github.com/makeitlal/preprocessed-beaver-matching`. We compare the MASCOT scheme with the Malicious Shamir scheme which uses the BGW protocol for multiplication. The results are shown below. Note that the results below assumes evaluation of the function using p parties, which is a generalization of the scenario presented above where $p = n$ is forced.

We find that using preprocessed Beaver triples shortens the online computation time by a large factor compared to the malicious Shamir scheme.

The empirical results align closely with the theoretical complexity analysis. Matrix multiplication requires $\Theta(mdn)$ multiplication gates, so both preprocessing cost and online latency scale approximately linearly with the number of funds m and feature dimension d when the remaining parameters are fixed.

For large m, both the online latency and the preprocessing time scales linearly. This is expected because both the number of required multiplications and comparisons used in f scales is linear in m. Tables 1 and 2 supports this, showing an approximately linear relation for large m in both tasks. An interesting consequence of this property is that preprocessed MASCOT will

p	m	Malicious Shamir	MASCOT (online latency + preprocessing)
5	5	0.069	0.024 + 3.179
	10	0.103	0.048 + 6.175
	20	0.154	0.043 + 11.796
	50	0.332	0.076 + 29.316
	100	0.640	0.129 + 59.189
10	5	0.237	0.074 + 13.903
	10	0.315	0.086 + 28.543
	20	0.474	0.113 + 49.546
	50	0.927	0.151 + 118.711
	100	1.851	0.233 + 240.066

Table 1: Latency runtime comparison (in seconds) for matrix multiplication with $n = 10$ and $d = 100$.

be almost as efficient even in the online setting of this problem where batches of inputs periodically arrive for computation rather than all at once. This indicates that this protocol is also applicable for settings where evaluation is expected to be sparse compared to the quiet period. In this case, parties can continuously preprocess during the quiet period, and switch to evaluation when batches of inputs arrive.

For large d, the online latency and the preprocessing time for the matrix multiplication task scales linearly. This is expected because the number of multiplications is ndm, which is linear in d. However, the argmax portion of f is independent of d, so the relation of the online latency and preprocessing time is approximately linear plus a constant. Table 3 supports this.

We additionally observe that preprocessing time grows rapidly with the number of parties p. This behavior is expected because MASCOT preprocessing requires pairwise oblivious transfer interactions between participating parties, causing communication volume to scale approximately quadratically in p. While preprocessing remains highly parallelizable, the total bandwidth and synchronization costs eventually dominate runtime for larger deployments. This suggests that the high preprocessing time will eventually not be worth the online computation speedup, as it is likely impractical for there to be sufficient preprocessing time just to carry out a few computations. For scenarios where a large number of parties is required, it may be more efficient to split the load of MPC across a few independent servers rather than all participating parties (but of course, new concerns such as the possible

p	m	Malicious Shamir	MASCOT (online latency + preprocessing)
5	5	0.161	0.074 + 9.993
	10	0.251	0.126 + 20.475
	20	0.439	0.149 + 38.648
	50	0.976	0.230 + 96.058
	100	1.874	0.398 + 190.272
10	5	0.507	0.241 + 38.519
	10	0.787	0.305 + 77.496
	20	1.318	0.361 + 158.903
	50	2.870	0.498 + 421.928
	100	5.699	0.961 + 858.034

Table 2: Latency runtime comparison (in seconds) for matrix multiplication with row-wise argmax with $n = 10$ and $d = 100$.

compromisation of servers arise).

7 Future Work

Our implementation of the preprocessed MASCOT scheme makes a couple of restrictions about the power of malicious parties. The first is that firms are not able to selectively abort the protocol, essentially forcing the protocol to stop due to the incompleteness of Beaver triple sharings. The second is that the funds are not allowed to submit false information, which can possibly be used to probe for the true values of the firms' preference vectors. These restrictions make our threat model more lenient than a threat model with active malicious parties with full capabilities.

Another topic of interest is whether the investors, which are the parties bearing the workload of MPC computation, are required to be fixed. A clear downside to preprocessing is that even if all parties are honest, switching parties often forces all previously preprocessed shares to be transferred from the party leaving the protocol to the party entering the protocol. Not only is this naive protocol implausible due to the high communication costs (at which point we would rather use malicious Shamir schemes which don't require preprocessing), but it also leaks the new party's Beaver triple shares. This could potentially be fixed if we relax the decentralization constraint, where there is a central system storing encryptions of the shares, which the parties

Task	d	Malicious Shamir	MASCOT (online latency + preprocessing)
MatMul	100	0.474	0.113 + 49.546
	200	0.783	0.163 + 106.506
	500	1.865	0.220 + 236.999
	1000	3.525	0.438 + 518.851
f	100	1.318	0.361 + 158.903
	200	1.617	0.395 + 208.594
	500	2.716	0.477 + 369.249
	1000	4.573	0.676 + 583.548

Table 3: Latency runtime comparison (in seconds) under varying feature vector dimensions d, with $p = n = 10$, and $m = 20$. For MASCOT, results are shown as online computation time + full preprocessing and online time.

can access whenever they need a Beaver triple. This is not possible under the assumption of an untrusted environment even if the central system is static, as the content of the system itself can possibly be compromised. However, if this issue is solved, we might consider applications of this protocol to online market settings where both buyers and sellers can enter the market if there is space available and leave the market once a match (such as a completed deal or transaction) is found.

We leave these potential areas of interest to future work.

8 Contributions

Most background research and code implementation was done by Evelyn.
Most slides, report writing, and correctness/security checks are done by Vasawat.
All parts of this project were verified by both before submission.

References

[1] Donald Beaver. Efficient multiparty protocols using circuit randomization. volume 576, pages 420–432, 08 1991.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*,

STOC '88, page 1–10, New York, NY, USA, 1988. Association for Computing Machinery.

[3] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. *IACR Cryptology ePrint Archive*, 2011:535, 01 2011.

[4] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, page 427–438, USA, 1987. IEEE Computer Society.

[5] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. Cryptology ePrint Archive, Paper 2016/505, 2016.

[6] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, page 129–140, Berlin, Heidelberg, 1991. Springer-Verlag.

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