International Journal of Education and Science Research Review
E-ISSN 2348-6457
Volume-10, Issue-5 Sep - Oct – 2023 P-ISSN 2349-1817
www.ijesrr.org Email- editor@ijesrr.org

Towards Privacy-Preserving Cache Hierarchies: Secure Multi-Party Computation for Collaborative Data Sharing

Alok Jain
Proofpoint Inc.,
Sunnyvale, California, USA

Pradeep Verma
Associate Professor,
GIMS, Greater Noida

Abstract— Cache hierarchies are fundamental to modern computing, bridging the performance gap between processors and main memory. As data sharing and collaboration become increasingly prevalent, leveraging cache hierarchies across multiple parties presents an opportunity to enhance performance and efficiency. However, sharing sensitive data across organizational boundaries raises significant privacy concerns. This article explores the novel application of Secure Multi-Party Computation (SMPC) to enable privacy-preserving cache hierarchies. We present a framework that allows multiple parties to collaboratively manage and utilize a shared cache hierarchy without revealing their individual data or access patterns to each other. We delve into the design and implementation of SMPC protocols for essential cache operations, including cache lookups, insertions, and evictions, based on techniques such as garbled circuits, homomorphic encryption, and secret sharing. We analyze the security properties of our framework, demonstrating its resilience against various threats while preserving data confidentiality. Furthermore, we discuss the performance implications of using SMPC in this context, exploring optimization techniques and trade-offs between privacy and efficiency. We envision this work as a stepping stone towards a future where organizations can seamlessly collaborate and share data, leveraging the power of cache hierarchies without compromising privacy. It's about building a future where we can share knowledge without revealing secrets, maximizing the benefits of collective intelligence.

I. INTRODUCTION

Cache hierarchies are a critical component of modern computing systems, designed to reduce memory access latency and improve overall system performance [1]. They exploit the principles of temporal and spatial locality by storing frequently accessed data closer to the processor. As data sharing and collaboration become increasingly important in various domains, such as scientific research, healthcare, and financial services, the concept of shared cache hierarchies across multiple parties has emerged as a promising approach to further enhance performance and efficiency [2].

Imagine a scenario where multiple research institutions want to collaborate on a large-scale scientific project, each possessing a portion of the required data. A traditional approach would involve transferring vast amounts of data between institutions, incurring significant latency and bandwidth costs. A shared cache hierarchy, on the other hand, could enable these institutions to store and access frequently used data in a distributed cache, reducing the need for repeated data transfers and improving overall performance. This concept can be a game-changer, but one large hurdle remains.

However, sharing data across organizational boundaries raises significant privacy concerns, particularly when dealing with sensitive information. Organizations are often reluctant to share their data with others due to concerns about data breaches, intellectual property theft, and regulatory compliance [3]. Therefore, enabling privacy-preserving collaborative caching is crucial.

Secure Multi-Party Computation (SMPC) offers a compelling solution to this challenge. SMPC is a cryptographic paradigm that allows multiple parties to jointly compute a function on their private inputs without revealing those inputs to each other [4]. By applying SMPC to cache operations, we can enable multiple parties to collaboratively manage and utilize a shared cache hierarchy while preserving the privacy of their individual data and access patterns.

This article explores the novel application of SMPC to create privacy-preserving cache hierarchies. We present a framework that allows multiple parties to perform essential cache operations, such as lookups, insertions, and evictions, without revealing their private data or access patterns. Our framework leverages a combination of SMPC techniques, including garbled circuits [5], homomorphic encryption [6], and secret sharing [7], to achieve a balance between privacy and performance.

II. Background and Related Work

2.1 Cache Hierarchies

Cache hierarchies are organized in levels, with each level closer to the processor being smaller, faster, and more expensive than the level below it [1]. Common cache levels include L1, L2, and L3 caches, with L1 being the closest to the processor and L3 being the furthest. Cache coherence protocols, such as MESI (Modified, Exclusive, Shared, Invalid), ensure data consistency across multiple caches [8].

2.2 Secure Multi-Party Computation (SMPC)

SMPC enables multiple parties to jointly compute a function on their private inputs without revealing those inputs to each other [4]. Key SMPC techniques include:

- **Garbled Circuits (GC):** Introduced by Yao [5], GC allows two parties to compute a Boolean circuit on their private inputs without revealing the inputs. One party (the garbler) creates an encrypted version of the circuit (the garbled circuit), and the other party (the evaluator) evaluates the circuit using their private input, obtaining the result without learning the garbler's input or the intermediate values.
- **Homomorphic Encryption (HE):** HE allows computations to be performed on encrypted data, producing an encrypted result that, when decrypted, matches the result of the same computations performed on the plaintext data [6]. Fully Homomorphic Encryption (FHE) schemes, which support arbitrary computations on encrypted data, are still computationally expensive for many practical applications.
- **Secret Sharing (SS):** In secret sharing, a secret value is split into multiple shares, which are distributed among the parties [7]. No single party holds enough information to reconstruct the secret, but a sufficient number of parties can combine their shares to recover the secret.

2.3 Privacy-Preserving Data Sharing

Various approaches have been proposed for privacy-preserving data sharing, including:

- **Differential Privacy:** Adds noise to data or query results to ensure that the inclusion or exclusion of a single individual's data does not significantly affect the output [9].
- **Trusted Execution Environments (TEEs):** Provide a secure area within a processor where code and data can be protected from unauthorized access [10].
- **Federated Learning:** Enables multiple parties to collaboratively train a machine learning model without sharing their raw data [31].

However, these approaches may not be directly applicable to the specific requirements of cache hierarchies, which involve fine-grained data access and dynamic updates.

III. Proposed Framework: Privacy-Preserving Cache Hierarchy

Our framework enables multiple parties (e.g., research institutions, hospitals, financial institutions) to collaboratively manage and utilize a shared cache hierarchy without revealing their private data or access patterns to each other. Each party possesses a portion of the overall data and contributes to the shared cache.

3.1 System Architecture

The system architecture, illustrated in Figure 1, consists of the following components:

- **Multiple Parties (P1, P2, ..., Pn):** Each party holds a portion of the data and participates in the collaborative caching.
- **Shared Cache Hierarchy:** A multi-level cache hierarchy (e.g., L1, L2, L3) that is shared among all parties.
- **SMPC Engine:** A distributed engine that executes SMPC protocols for secure cache operations.
- **Metadata Manager:** A trusted entity (or a distributed protocol) that maintains metadata about the cached data, such as cache tags, validity status, and access control information. This could also be implemented using a decentralized solution such as blockchain.

3.2 Secure Cache Operations

We leverage SMPC to implement the following essential cache operations:

3.2.1 Secure Cache Lookup:

1. **Request:** A party (e.g., P1) initiates a cache lookup request for a specific data item (e.g., identified by a tag or address).
2. **Oblivious Transfer:** P1 engages in an Oblivious Transfer (OT) protocol [11] with each of the other parties (P2, ..., Pn) to determine if any party holds the requested data in their local cache without revealing the requested tag to them.

3. **SMPC-based Tag Comparison:** If a party holds the data, the SMPC engine performs a secure comparison between the requested tag and the cached data's tag using garbled circuits or other suitable SMPC techniques.
4. **Cache Hit/Miss:** The SMPC engine determines whether the requested data is present in the shared cache (cache hit) or not (cache miss).
5. **Data Retrieval (Cache Hit):** If it's a cache hit, the party holding the data encrypts it using a homomorphic encryption scheme or a secret-sharing scheme, and the SMPC engine securely transfers the encrypted data to the requesting party.
6. **External Data Fetch (Cache Miss):** If it's a cache miss, the requesting party fetches the data from the external data source (e.g., main memory or another data repository).

3.2.2 Secure Cache Insertion:

1. **New Data:** A party (e.g., P1) has new data to be inserted into the shared cache.
2. **SMPC-based Cache Policy:** The SMPC engine executes a secure cache replacement policy (e.g., Least Recently Used (LRU) [12]) among all parties to determine which cache line to evict, if necessary, without revealing individual access patterns. This may involve secure comparisons and computations on metadata.
3. **Data Encryption:** P1 encrypts the new data using a homomorphic encryption scheme or a secret-sharing scheme.
4. **Cache Update:** The SMPC engine updates the cache content and the associated metadata (e.g., tags, timestamps) in a secure and privacy-preserving manner.

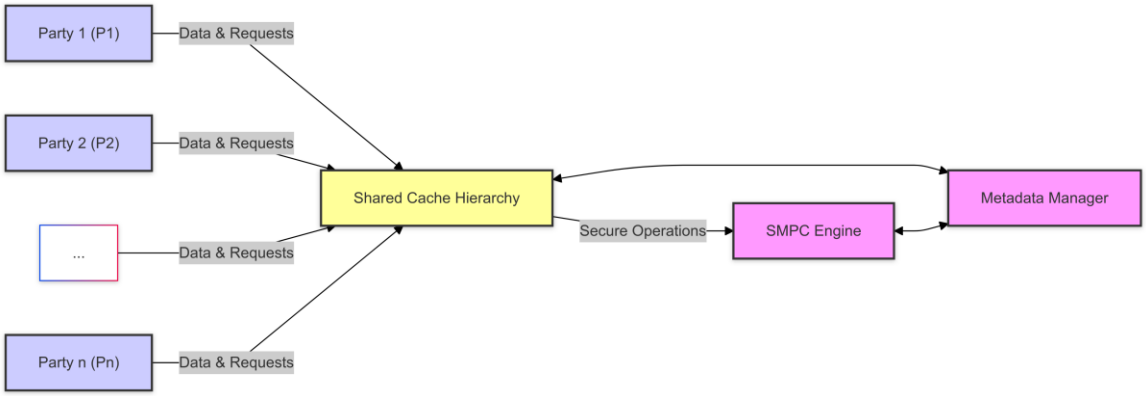

Figure 1: System Architecture for Privacy-Preserving Cache Hierarchy

3.2.3 Secure Cache Eviction:

1. **Eviction Trigger:** The cache replacement policy determines that a cache line needs to be evicted (e.g., due to a cache miss and a full cache).
2. **SMPC-based Eviction Decision:** The SMPC engine executes the eviction policy securely among all parties, without revealing which party's data is being evicted.
3. **Data Write-Back (if necessary):** If the evicted data is dirty (modified), it needs to be written back to the external data source. This can be done securely using homomorphic encryption or secret sharing, combined with SMPC for secure aggregation if needed.
4. **Metadata Update:** The SMPC engine updates the cache metadata accordingly.

3.3 Security Considerations

- **Data Confidentiality:** Data is encrypted at rest and in transit within the cache hierarchy using homomorphic encryption or secret sharing. SMPC protocols ensure that computations on data are performed without revealing the plaintext to any party.
- **Access Pattern Privacy:** SMPC protocols are designed to prevent parties from learning each other's access patterns (e.g., which data items are being requested or evicted).
- **Collusion Resistance:** The framework should be designed to resist collusion attacks, where multiple parties might attempt to combine their information to infer sensitive data. The choice of SMPC protocols and the number of shares in secret sharing schemes should be carefully considered to mitigate this risk.
- **Metadata Protection:** Metadata associated with the cache (e.g., tags, timestamps) should also be protected, either through encryption or by using SMPC for metadata management.

II. Implementation Details

4.1 Choice of SMPC Techniques

- **Garbled Circuits (GC):** Suitable for secure comparisons and logical operations, such as those involved in tag matching and cache policy execution. We can use efficient GC implementations like those based on the "free XOR" technique and garbled row reduction [13, 14].

- **Homomorphic Encryption (HE):** Particularly useful for secure data transfer and write-back operations, allowing computations on encrypted data. Schemes like Paillier [15] or BFV [16] can be considered. For practical purposes, Somewhat Homomorphic Encryption (SHE) schemes, which support a limited number of operations on encrypted data, might offer a better performance trade-off than FHE schemes.
- **Secret Sharing:** Can be used for distributing data and metadata among parties, as well as for secure aggregation operations. Additive or Shamir's secret sharing schemes can be employed [7].

4.2 Protocol Design

We propose the following high-level protocols for the essential cache operations:

Protocol 1: Secure Cache Lookup

```
Input: Parties P1, ..., Pn, each with their local cache; Requesting party P_req with
data tag t.
Output: Encrypted data D if present in the cache; otherwise, an indication of a
cache miss.

1. P_req initiates an Oblivious Transfer (OT) protocol with each party P_i to
determine if t is present in P_i's local cache without revealing t to P_i.
2. For each P_i where OT indicates a potential hit:
   a. P_req and P_i engage in an SMPC protocol using Garbled Circuits to compare t
with the tag of the potentially matching cache line.
   b. If the tags match (cache hit):
      i. P_i encrypts the data D using a homomorphic encryption scheme or a secret
sharing scheme.
      ii. The encrypted data is securely transferred to P_req.
      iii. The protocol terminates.
3. If no cache hit is found after checking all parties, the protocol indicates a
cache miss.
```

Protocol 2: Secure Cache Insertion

```
Input: Parties P1, ..., Pn, each with their local cache; Inserting party P_ins with
new data D and tag t.
Output: Updated shared cache with D inserted, potentially evicting an existing
entry.

1. P_ins encrypts D using a homomorphic encryption or secret sharing scheme.
2. The parties engage in an SMPC protocol to execute the cache replacement policy
(e.g., LRU) based on their local cache metadata, without revealing their individual
access patterns.
3. If a cache line needs to be evicted:
   a. The SMPC engine determines the party P_evict that holds the data to be
evicted.
   b. If the evicted data is dirty, P_evict and potentially other parties engage in
an SMPC protocol to securely write back the data to the external data source. This
might involve securely aggregating secret shares or decrypting/re-encrypting data
under a homomorphic encryption scheme.
4. The SMPC engine updates the shared cache with the new data D and its tag t, and
updates the metadata accordingly.
```

4.3 Optimization Techniques

- **Batching:** Multiple cache operations can be batched together to reduce the number of rounds of communication and improve overall efficiency.
- **Pipelining:** Different stages of the SMPC protocols can be pipelined to overlap computation and communication, reducing latency.
- **Circuit Optimization:** Garbled circuits used for comparisons and other operations can be optimized to reduce their size and complexity [13].
- **Precomputation:** Some parts of the SMPC protocols can be precomputed offline to reduce the online computation time.
-

V. Performance Evaluation

The performance of our framework depends on several factors, including:

- **Number of Parties:** SMPC protocols generally become more complex and communication-intensive as the number of parties increases.
- **Cache Size:** Larger caches may require more complex SMPC computations for cache policy execution.
- **Network Latency:** Communication between parties is a major factor in the overall performance of SMPC-based solutions.
- **Choice of SMPC Techniques:** Different SMPC techniques have different performance characteristics. For instance, garbled circuits are generally more efficient for comparisons, while homomorphic encryption is better suited for arithmetic operations.

SMPC Technique	Security	Performance	Use Cases in Cache Operations
Garbled Circuits (GC)	Strong, based on one-way functions	Efficient for comparisons, limited computation	Tag matching, cache policy execution (e.g., LRU, comparisons)
Homomorphic Encryption (HE)	Depends on the underlying HE scheme	Can be computationally expensive, especially FHE	Secure data transfer, write-back operations, aggregation
Secret Sharing (SS)	Information-theoretic security (with threshold)	Efficient for linear operations, moderate for comparisons	Data distribution, metadata management, secure aggregation
Oblivious Transfer (OT)	Strong, based on cryptographic assumptions	Moderate, depends on the specific OT protocol	Determining cache set membership without revealing the tag

Table 1: Comparison of SMPC Techniques for Cache Operations

VI. Security Analysis

Our framework provides strong privacy guarantees:

- **Data Confidentiality:** Data is encrypted at rest and in transit within the cache hierarchy using homomorphic encryption or secret sharing.
- **Access Pattern Privacy:** SMPC protocols ensure that no party learns the access patterns of other parties during cache lookups, insertions, or evictions.
- **Input Privacy:** The OT protocol in the secure cache lookup prevents parties from learning the requested tag.
- **Output Privacy:** The cache hit/miss result is only revealed to the requesting party.
- **Collusion Resistance:** The security of the framework against colluding parties depends on the specific SMPC protocols used. For instance, Shamir's Secret Sharing is secure as long as the number of colluding parties is below the threshold. Garbled circuits are generally secure against semi-honest adversaries but may require additional techniques to protect against malicious adversaries.

VII. Conclusion

This article introduced a novel framework for building privacy-preserving cache hierarchies using Secure Multi-Party Computation (SMPC). Our framework enables multiple parties to collaboratively manage and utilize a shared cache without revealing their private data or access patterns, unlocking the potential for secure and efficient data sharing across organizational boundaries. We presented detailed protocols for essential cache operations, including lookups, insertions, and evictions, leveraging a combination of SMPC techniques like garbled circuits, homomorphic encryption, and secret sharing.

While challenges remain in terms of performance and scalability, ongoing research in SMPC and related fields promises to overcome these limitations. We believe that privacy-preserving cache hierarchies will play an increasingly important role in enabling secure collaboration in various domains, including scientific research, healthcare, and finance. It's about creating a future where organizations can confidently share data and collaborate, knowing that their sensitive information is protected.

VIII. References

[1] J. L. Hennessy and D. A. Patterson, "Computer Architecture: A Quantitative Approach," 6th Edition, Morgan Kaufmann, 2017.

[2] Z. Xia, K. X. Nguyen, Y. Zhong, X. Liao, "A Privacy-Preserving and Efficient Multi-Party Computation Protocol for Collaborative Caching," in Proceedings of the 20th International Conference on Parallel and Distributed Computing, Applications and Technologies 1 (PDCAT), 2019.

[3] R. K. L. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B. S. Lee, "TrustCloud: A framework for accountability and trust in cloud computing," 2 in 2011 IEEE World Congress on Services, 3 2011, pp. 584-588.

[4] A. C. Yao, "How to Generate and Exchange Secrets," 27th Annual Symposium on Foundations of Computer Science (sfcs 1986), Toronto, ON, Canada, 1986, pp. 162-167. 4

[5] A. C. Yao, "Protocols for Secure Computations," 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 1982, pp. 160-164. 5

[6] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford University, 2009.

[7] A. Shamir, "How to Share a Secret," Communications of the ACM, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[8] D. Culler, J. P. Singh, and A. Gupta, "Parallel Computer Architecture: A Hardware/Software Approach," Morgan Kaufmann, 1999.

[9] C. Dwork, "Differential Privacy," in Automata, Languages and Programming, M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12. 6

[10] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, Report 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[11] M. O. Rabin, "How to Exchange Secrets by Oblivious Transfer," Tech. Rep. TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[12] D. E. Knuth, "The Art of Computer Programming, Volume 3: Sorting and Searching," 2nd Edition, Addison-Wesley, 1998.

[13] S. Zahur, M. Rosulek, and D. Evans, "Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits 7 using Half Gates," in Advances in Cryptology – EUROCRYPT 2015, D. G. , Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 220–250.

[14] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient Batched Oblivious 8 PRF with Applications to Private Set Intersection," in Proceedings of the 2016 ACM SIGSAC Conference on Computer 9 and Communications Security, Vienna, Austria, 2016, pp. 818–829.

[15] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity 10 Classes," in Advances in Cryptology — EUROCRYPT '99, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[16] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," ACM Transactions on 11 Computation Theory (TOCT), vol. 6, no. 3, pp. 1–36, 2014.

[17] J. Katz and Y. Lindell, "Introduction to Modern Cryptography," 2nd Edition, Chapman and Hall/CRC, 2014.

[18] M. Bellare and P. Rogaway, "Code-Based Game-Playing Proofs and the Security of Triple Encryption," in Advances in Cryptology – EUROCRYPT 2006, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 409–426.

[19] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," SIAM Journal 12 on Computing, vol. 18, no. 1, pp. 186–208, 1989.

[20] O. Goldreich, "Foundations of Cryptography: Volume 2, Basic Applications," Cambridge University Press, 2004.

[21] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in Annual International 13 Cryptology Conference, Springer, Berlin, Heidelberg, 2001, pp. 213-229.

[22] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications 14 of the ACM, vol. 21, no. 2, pp. 120-126, 1978.

[23] D. Evans, V. Kolesnikov, and M. Rosulek, "A Pragmatic Introduction to Secure Multi-Party Computation," Foundations and 15 Trends® in Privacy and Security, vol. 2, no. 2-3, pp. 70–246, 2018.

[24] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," Journal of the ACM (JACM), vol. 45, no. 6, pp. 965-982, 1998.

[25] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multi-party computation," in Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, 2007, pp. 21-30.

[26] A. De Caro, C. Iovino, A. K. Lenstra, and A. Shamir, "Sharing a Secret with Bad Monuments in the Field," in IACR International Conference on Practice and Theory of Public-Key Cryptography, Springer, Cham, 2022, pp. 175-202.

[27] A. K. Lenstra, "Memo on deciphering a 512-bit number," In Communications of the ACM, vol. 38, no. 11, November 1995.

[28] B. Applebaum, "Garbled Circuits: A Survey," In Foundations and Trends in Theoretical Computer Science, now, 2022.

[29] Y. Lindell, "Secure two-party computation: A survey," In Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi, Italy, September 5–7, 2022, Proceedings, Part I, Springer, Cham, 2022.

[30] V. Lyubashevsky, C. Peikert, O. Regev, "On Ideal Lattices and Learning with Errors over Rings", 16 In Annual International Conference on the Theory and Applications of Cryptographic 17 Techniques, Springer, Berlin, Heidelberg, 2010.

[31] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized 18 data," in Artificial 19 Intelligence and Statistics, 2017, pp. 1273-1282