A Retrospective on Path ORAM

Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas

***Abstract*—Path oblivious RAM (ORAM) is an ORAM protocol that simultaneously enjoys simplicity and efficiency. As a result, it holds promise to provide cryptographic-grade and practical access pattern protection in multiple application domains, including but not limited to secure hardware. In this paper, we review Path ORAM's key ideas and contribution, summarize its impact and subsequent works, and discuss future directions.**

***Index Terms*—Computer security, cryptographic protocols, microprocessors, side-channel attacks.**

I. Introduction

IMAGINE that a client would like to outsource a database of sensitive records to an untrusted server. To protect the privacy of the data, the client encrypts all records before uploading them to the server. However, it is well-known that encryption alone is not sufficient for ensuring privacy; numerous works have shown that by observing the access patterns alone, very sensitive information can be reconstructed [7], [34]–[36], [65], [71], [76]. As a simple example, if the client is performing binary searches on sorted records, the sequence of memory accesses reveals whether a small or large key is being searched.

Therefore, the client would like to "encrypt" the access patterns as well. The ground-breaking works by Goldreich *et al.* [26], [27] introduced oblivious RAM (ORAM), an algorithm that obfuscates access patterns to data such that no information is leaked. Their ingenious construction accomplishes this by permutating (encrypted) data blocks stored on the server and periodic reshuffling data around. Their construction incurs $O(\log^3 N)$ bandwidth overhead. Throughout this paper, N denotes the total number of logical blocks.

Goldreich and Ostrovsky's work was an amazing theoretical breakthrough. At the same time, it raised major questions that have intrigued the community since. On the theoretical front, a lower bound was shown by Goldreich and Ostrovsky [27], and recently strengthened by Larsen and Nielsen [37], that any ORAM scheme must suffer from $\Omega(\log N)$ bandwidth overhead. *Can we bridge the gap between upper and lower bounds in bandwidth?* On the practical side, Goldreich and Ostrovksy's construction would have incurred $10^5\times\sim10^6\times$ overhead in practice for a medium-sized database (e.g., one that contains millions of blocks). *Can we ever hope to get ORAMs adopted in practice?*

The main technical tool behind Goldreich and Ostrovsky's construction is for the client to carefully maintain (on the server) a hierarchy of data structures of geometrically growing sizes—for this reason, Goldreich and Ostrovsky's paradigm is commonly referred to as "hierarchical ORAMs." Since then, several subsequent works [28], [69], [70] improved hierachical ORAMs.

It was not until 2011 that a fundamentally new paradigm for constructing ORAMs, commonly referred to as the *tree-based* framework, was proposed in a work by Shi *et al.* [58]. The tree-based paradigm captured the community's attention primarily due to its conceptual simplicity. Although Shi *et al.*'s [58] initial tree-based ORAM still suffers from $O(\log^3 N)$ overhead, a line of subsequent works have quickly improved the asymptotical and practical performance of tree-based ORAMs [10], [13], [23], [53], [63]. Among these works, Path ORAM [63] is arguably the most well-known.

With an extremely simple construction that can be described in only 15 lines of pseudo-code (Fig. 1), Path ORAM made a leap forward in answering the above open challenges.

1) On the practical side, Path ORAM or its close variants have become the scheme of choice in multiple research communities, including secure hardware, cloud computing, and secure multiparty computation.
2) On the theoretical front, Path ORAM was the first small client storage scheme to match the logarithmic lower bound for some parameter regimes.
3) Last but not least, Path ORAM is also an ideal candidate for learning ORAM and for pedagogy.

Organization: Section II reviews the Path ORAM scheme and result. We will summarize the broad applications and impact of Path ORAM in Section III, and then end this paper by discussing future directions.

II. Path ORAM Review

A. ORAM Definition

An ORAM scheme is an algorithm running on a trusted client that translates the client's logical memory requests to a sequence of accesses to an untrusted storage server. Depending on the application scenario, the client may be a CPU, an

Manuscript received January 22, 2019; revised April 22, 2019; accepted June 8, 2019. Date of publication June 27, 2019; date of current version July 17, 2020. This paper was recommended by Associate Editor J. Rajendran. *(Corresponding author: Ling Ren.)*

E. Stefanov was with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720 USA.

M. van Dijk is with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269 USA.

E. Shi is with the Department of Computer Science, Cornell University, Ithaca, NY 14853 USA.

C. Fletcher and L. Ren are with the Department of Computer Science, University of Illinois Urbana–Champaign, Champaign, IL 61801 USA (e-mail: renling@illinois.edu).

X. Yu and S. Devadas are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Digital Object Identifier 10.1109/TCAD.2019.2925398

enclave, or a user of some cloud storage service. An ORAM scheme provides a standard memory abstraction where a client can make two types of requests of the form (read, a, $\perp$) or (write, a, data*): both instructions would return the current contents residing at logical address a and the latter additionally updates the contents at a with data*. An ORAM scheme is said to be secure if for any two logical request sequences of the same length $\mathbf{I}_0$ and $\mathbf{I}_1$, the physical access patterns produced by the scheme (i.e., the sequence of physical locations accessed) are indistinguishable. Note that this definition implies that the physical access patterns do not reveal the logical address, the operation (read versus write), or the data of any request.

B. Protocol

Path ORAM follows the tree-based paradigm of Shi *et al.* [58] and organizes untrusted server-side storage as a binary tree of height $L = O(\log N)$. Each node in the tree is a bucket that has $Z = O(1)$ slots to store logical data blocks. Every slot stores either a *real* block or a *dummy* block. Every time a block is updated on the server, the block is re-encrypted by the client using fresh randomness.

Each block is assigned to a path in the binary tree. Path ORAM maintains the following key invariant: *At all times, each data block occupies a slot in some bucket on the path it is assigned to, or is stored locally in a client-side data structure called the stash.*

To aid understanding, we shall first make a simplifying assumption that the client locally stores the block-to-path mapping for all blocks—this auxiliary data structure is called a *position map*. We later discuss how to remove this assumption.

With the above invariant, reading/writing a block in Path ORAM is done through the algorithm in Fig. 1. To elaborate, it consists of the following steps.

1) The client looks up the position map to determine the path that the block is mapped to (line 1).
2) The client downloads all blocks in all buckets along that path, decrypts those blocks and adds them to the stash (lines 2–4). By the invariant, the block of interest now lives in the stash. The client can now update the block's data if the operation was a write (lines 5 and 6).
3) The client flips coins to assign a new random path to the accessed block by updating the position map (lines 7 and 8).
4) The client re-encrypts and uploads as many blocks as possible in the stash back to the binary tree, along the same path the client read in Step 2, *placing the blocks as close to the leaf nodes as possible* while still respecting the invariant (lines 9–14).

Security, Correctness, and Efficiency: In the above protocol, every read or write request requires the client to download and upload a random path. The choice of the random path was determined when the block was last accessed and has not been revealed to the server prior to this access. Thus, security of the protocol is easy to see.

The concrete bandwidth overhead (without recursion) is exactly $2(L+1)Z = \Theta(\log N)$. The Path ORAM

Access(op, a, data*):

1: $x \leftarrow$ position[a]
2: **for** $\ell \in \{0, 1, \ldots, L\}$ **do**
3: $\quad S \leftarrow S \cup$ DownloadBucket($\mathcal{P}(x, \ell)$)
4: **end for**
5: data $\leftarrow$ Read block a from S
6: data* $\leftarrow$ (op = write) ? data* : data
7: position[a] $\leftarrow x^* \leftarrow$ UniformRandom$(0 \ldots 2^L - 1)$
8: $S \leftarrow (S - \{(\mathsf{a}, x, \mathsf{data})\}) \cup \{(\mathsf{a}, x^*, \mathsf{data}^*)\}$
9: **for** $\ell \in \{L, L-1, \ldots, 0\}$ **do**
10: $\quad S' \leftarrow \{(\mathsf{a}', x', \mathsf{data}') \in S : \mathcal{P}(x, \ell) = \mathcal{P}(x', \ell)\}$
11: $\quad S' \leftarrow$ Select $\min(|S'|, Z)$ blocks from S'.
12: $\quad S \leftarrow S - S'$
13: $\quad$ UploadBucket($\mathcal{P}(x, \ell), S'$)
14: **end for**
15: **return** data

Fig. 1. Protocol for Path ORAM access. Read or write a data block identified by a. If op = read, the input parameter data* = None, and the Access operation reads block a from the ORAM. If op = write, the Access operation writes the specified data* to the block identified by a and returns the block's old data. position is the position map. S is the stash. L is the depth of the tree. $\mathcal{P}(x, \ell)$ is the ℓth bucket (counting from root to leaf) on path x. Z is the number of slots per bucket.

papers [60], [63] provided a stochastic analysis that proved the following result: As long as the bucket size $Z \geq 5$ and the tree height $L \geq \lceil \log N \rceil$, then over T number of accesses, the stash size never exceeds R except with $T \cdot \exp(-\Omega(R))$ probability. Experiments show that $Z = 4$ with $L = \lceil \log N \rceil - 1$ works well in practice. Further decreasing Z or L would require additional mechanisms and new analysis to make sure the stash remains small [55].

Recursion: One remaining issue is the position map size, which has $\Theta(N)$ entries (one for each block). This structure can be shrunk to as small as a constant number of entries by applying the ORAM recursion trick proposed in Shi *et al.* [58]. The idea is to store the original position map in a second Path ORAM. The position map of this second ORAM is smaller than the original one. This process can be repeated until the position map becomes small enough to fit in client storage. We remark that, in order to get the aforementioned $O(\log N)$ bandwidth overhead, it is crucial to use $\Theta(\log N)$-bit blocks for position map ORAMs, regardless of the data block size B. With this parametrization suggested in the journal version [60], Path ORAM consumes a bandwidth of $O(B \log N + \log^3 N)$ measured in bits, which is $O(B \log N)$ if $B = \Omega(\log^2 N)$. The conference version [63] did not adopt this parametrization and had suboptimal bandwidth.

III. Impact and Subsequent Works

Since its proposal, Path ORAM has had significant impact and spurred follow-up research. In this section, we review recent works that built on Path ORAM.

A. Algorithmic Improvements

Many subsequent ORAM algorithms followed the Path ORAM paradigm. Ren *et al.* [53] presented a variant of Path

ORAM with improved bandwidth and simpler stash analysis. Wang *et al.* [66] presented Circuit ORAM, which matches Path ORAM's asymptotic $O(\log N)$ bandwidth and improves client storage to $O(1)$. Onodera and Shibuya [50] presented techniques to reduce Path ORAM's server storage from $O(N)$ to $N + o(N)$. Boyle *et al.* [5], Chen *et al.* [12], and Chan and Shi [10] studied *parallel* ORAMs and adopt Path ORAM or its close variants as starting points of their constructions. Wang *et al.* [68] showed how to construct efficient oblivious data structures by piggybacking on Path ORAM's recursion.

B. Outsourced Storage and Databases

Bindschaedler *et al.* [4] and Chang *et al.* [11] conducted experiments comparing ORAM schemes for practical outsourced storage. Both papers conclude that Path ORAM is the most efficient small client storage scheme but has worse bandwidth than partition ORAM [62], the most efficient scheme with large client storage. However, neither work was aware of our improved variant of Path ORAM [53]. This variant utilizes client storage more effectively and we expect it to match the bandwidth of partition ORAM for storage outsourcing [62]. Crooks *et al.* [15] developed a cloud-based oblivious key-value store system based on this improved variant.

Bindschaedler *et al.* [4] also identified lack of support for concurrent accesses as a major bottleneck in existing ORAM schemes. This issue has since been partially addressed by the works of Sahin *et al.* [56] and Chakraborti and Sion [8], which extend Path ORAM to support concurrent accesses from many clients. Another work by Chakraborti *et al.* [9] extends Path ORAM to support range queries in a locality-friendly manner.

C. Secure Hardware

When client storage is limited, as is the case for secure hardware, Path ORAM is by far the most efficient ORAM. It was embraced by secure hardware designs immediately after its invention. The Ascend secure processor [20], [54], the Phantom secure processor [43], the GhostRider system [38], and the practical obfuscation project [48] all adopt Path ORAM. A number of works suggested improvements to Path ORAM from a hardware, system or architectural level [21], [22], [49], [64], [73]–[75]. To the best of our knowledge, Path ORAM is the only ORAM to have been implemented in hardware. Most other ORAMs are difficult to implement even in software.

The Ascend secure processor built a custom silicon implementation of Path ORAM with the following specifications [54]. The Path ORAM controller consumes 0.51 mm^2 area in 32 nm SOI process. The power consumption is about 30 mW at 250 MHz, 75 mW at 500 MHz, 150 mW at 750 MHz, and 300 mW at 857 MHz. One access of 512-bit data (one cache line) from main memory has a latency of around 1275 cycles, which leads to an average slowdown of about 4× on SPEC INT 2006 benchmarks according to simulation [21].

D. Secure Computation

ORAM has been adopted as a building block for secure computation in the RAM model to avoid converting RAM programs into circuits [29]. Keller and Scholl [24] and Gentry *et al.* [25] used Path ORAM as the underlying ORAM scheme. Wang *et al.* devised ORAM schemes specifically optimized for secure computation [66], [67] following the Path ORAM paradigm.

E. Data Oblivious Computing

Path ORAM has taken on similar roles in many data oblivious programming proposals, such as data oblivious ISA extensions [72], ZeroTrace [57], and Oblix [46]. A line of work [40], [41] designs domain-specific programming languages and compilers that automatically compile a program into an oblivious format, which may then be executed by either trusted hardware or secure multiparty computation protocols. Type systems have also been developed to ensure that the source or target program indeed satisfies obliviousness [16], [38], [39].

IV. Future Directions

We have mentioned that Path ORAM is the first small client storage ORAM to achieve the asymptotically optimal $O(\log N)$ bandwidth; but we noted that it holds only when the block size $B = \Omega(\log^2 N)$. Recently, Asharov *et al.* [2], building on Patel *et al.* [52], has proposed a small client ORAM scheme that achieves $O(\log N)$ bandwidth with $B = \Omega(\log N)$ block size. Since it takes $\Theta(\log N)$ bits to address a logical memory with N blocks, $B = \Theta(\log N)$ is the minimum block size worth catering for. In this sense, Asharov *et al.* [2] claimed to have achieved asymptotic optimality in the standard model. Currently, their construction is highly complex, has huge hidden constants, and fundamentally relies on the hierarchical ORAM framework [27]. It is interesting to investigate whether the same result can be achieved with simpler constructions, possibly following the tree-based (Path) ORAM paradigm.

Since ORAM schemes are approaching the lower bound in the standard model, it is important to explore new models in which overhead can be further reduced. For example, several works have studied how untrusted computation at the storage can dramatically reduce bandwidth costs [17], [18], [45], [53]. This is a natural model in storage outsourcing and secure computation. It can be relevant to hardware and embedded security as well since a storage device starts to support a full software stack and even main memory is being endowed with local computation (e.g., hybrid memory cubes). Onion ORAM [18] in particular shows how untrusted computation can be used to "break" the $O(\log N)$ bandwidth lower bound and achieve $O(1)$ bandwidth overhead. Unfortunately, Onion ORAM requires expensive untrusted computation of homomorphic encryption and an open challenge is how to achieve similar results with cheaper computation.

Another active line of research considers a multiserver setting and assumes no collusion or at least one honest server [51], [61]. Recent works in this direction incorporate

ideas from untrusted computation to further improve efficiency [1], [31]. A related line of work observes that there is no client versus server distinction when using ORAM in secure computation and these schemes are known as distributed ORAMs [6], [19], [42].

In large part due to Path ORAM, we are at an exciting juncture in ORAM's history where bandwidth overhead has been reduced to the point where ORAM is now practical in many settings today. One major direction for future work is therefore to explore new applications where ORAM can be used, such as secure enclaves [14], [33] and symmetric searchable encryption [32], [47], [59].

As ORAM finds more applications, we will need to solve new usability challenges. For example, how can we anonymously share a single ORAM across multiple distrusting users and/or coordinate users with complex access controls? There have been attempts in these directions [3], [30], [44] but much future research is needed to get a practical solution.

REFERENCES

[1] I. Abraham, C. W. Fletcher, K. Nayak, B. Pinkas, and L. Ren, "Asymptotically tight bounds for composing ORAM with PIR," in *Proc. IACR Int. Workshop Public Key Cryptography*, 2017, pp. 91–120.

[2] G. Asharov, I. Komargodski, W.-K. Lin, K. Nayak, and E. Shi, "OptORAMa: Optimal oblivious RAM," Cryptol. ePrint Archive, Rep. 2018/892, 2018.

[3] M. Backes, A. Herzberg, A. Kate, and I. Pryvalov, "Anonymous RAM," in *Proc. Eur. Symp. Res. Comput. Security*, 2016, pp. 344–362.

[4] V. Bindschaedler, M. Naveed, X. Pan, X. Wang, and Y. Huang, "Practicing oblivious access on cloud storage: The gap, the fallacy, and the new way forward," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 837–849.

[5] E. Boyle, K.-M. Chung, and R. Pass, "Oblivious parallel RAM and applications," in *Proc. Theory Cryptography Conf.*, 2016, pp. 175–204.

[6] P. Bunn, J. Katz, E. Kushilevitz, and R. Ostrovsky, "Efficient 3-party distributed ORAM," Cryptol. ePrint Archive, Rep. 2018/706, 2018.

[7] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 668–679.

[8] A. Chakraborti and R. Sion, "ConcurORAM: High-throughput stateless parallel multi-client ORAM," in *Proc. Netw. Distrib. Syst. Security*, 2019, pp. 1–15.

[9] A. Chakraborti, A. J. Aviv, S. G. Choi, T. Mayberry, D. S. Roche, and R. Sion "rORAM: Efficient range ORAM with $O(\log^2 N)$ locality," in *Proc. Netw. Distrib. Syst. Security*, 2019, pp. 1–15.

[10] T.-H. H. Chan and E. Shi, "Circuit OPRAM: Unifying statistically and computationally secure ORAMs and OPRAMs," in *Proc. Theory Cryptography Conf.*, 2017, pp. 72–107.

[11] Z. Chang, D. Xie, and F. Li, "Oblivious RAM: A dissection and experimental evaluation," *Proc. VLDB Endow.*, vol. 9, no. 12, pp. 1113–1124, 2016.

[12] B. Chen, H. Lin, and S. Tessaro, "Oblivious parallel RAM: Improved efficiency and generic constructions," in *Proc. Theory Cryptography Conf.*, 2016, pp. 205–234.

[13] K. Chung, Z. Liu, and R. Pass, "Statistically-secure ORAM with $\tilde{o}(\log^2 n)$ overhead," in *Proc. 20th Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2014, pp. 62–81.

[14] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *Proc. USENIX Security*, 2016, pp. 857–874.

[15] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi, "Obladi: Oblivious serializable transactions in the cloud," in *Proc. 13th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 727–743.

[16] D. Darais, C. Liu, I. Sweet, and M. Hicks, "A language for probabilistically oblivious computation," *CoRR*, vol. abs/1711.09305, 2017.

[17] J. L. Dautrich, Jr., E. Stefanov, and E. Shi, "Burst ORAM: Minimizing ORAM response times for bursty access patterns," in *Proc. USENIX Security Symp.*, 2014, pp. 749–764.

[18] S. Devadas, M. van Dijk, C. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion ORAM: A constant bandwidth blowup oblivious RAM," in *Proc. Theory Cryptography Conf.*, 2016, pp. 145–174.

[19] S. Faber, S. Jarecki, S. Kentros, and B. Wei, "Three-party ORAM for secure computation," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2015, pp. 360–385.

[20] C. Fletcher, M. V. Dijk, and S. Devadas, "A secure processor architecture for encrypted computation on untrusted programs," in *Proc. 7th ACM Workshop Scalable Trusted Comput.*, 2012, pp. 3–8.

[21] C. W. Fletcher, L. Ren, A. Kwon, M. van Dijk, and S. Devadas, "Freecursive ORAM: [Nearly] free recursion and integrity verification for position-based oblivious RAM," in *Proc. ACM SIGARCH Comput. Archit. News*, vol. 43, 2015, pp. 103–116.

[22] C. Fletcher *et al.*, "A low-latency, low-area hardware oblivious RAM controller," in *Proc. IEEE 23rd Annu. Int. Symp. Field Program. Custom Comput. Mach. (FCCM)*, 2015, pp. 215–222.

[23] C. Gentry, K. A. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs, "Optimizing ORAM and using it efficiently for secure computation," in *Proc. Int. Symp. Privacy Enhanc. Technol. Symp.*, 2013, pp. 1–18.

[24] M. Keller and P. Scholl, "Efficient, oblivious data structures for MPC," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2014, pp. 506–525.

[25] C. Gentry, S. Halevi, C. Jutla, and M. Raykova, "Private database access with HE-over-ORAM architecture," in *Proc. Int. Conf. Appl. Cryptography Netw. Security*, 2015, pp. 172–191.

[26] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *Proc. 9th Annu. ACM Symp. Theory Comput.*, 1987, pp. 182–194.

[27] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[28] M. T. Goodrich and M. Mitzenmacher, "Privacy-preserving access of outsourced data via oblivious RAM simulation," in *Proc. Int. Colloquium Automata Lang. Program. (ICALP)*, 2011, pp. 576–587.

[29] S. D. Gordon *et al.*, "Secure two-party computation in sublinear (amortized) time," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 513–524.

[30] A. Hamlin, R. Ostrovsky, M. Weiss, and D. Wichs, "Private anonymous data access," Cryptol. ePrint Archive, Rep. 2018/363, 2018.

[31] T. Hoang, C. D. Ozkaptan, A. A. Yavuz, J. Guajardo, and T. Nguyen, "S3ORAM: A computation-efficient and constant client bandwidth blowup ORAM with Shamir secret sharing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 491–505.

[32] T. Hoang, A. A. Yavuz, F. B. Durak, and J. Guajardo, "Oblivious dynamic searchable encryption on distributed cloud systems," in *Proc. IFIP Annu. Conf. Data Appl. Security Privacy*, 2018, pp. 113–130.

[33] Intel. (2013). *Intel Software Guard Extensions Programming Reference*. [Online]. Available: https://software.intel.com/sites/default/files/329298-001.pdf

[34] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *Proc. Netw. Distrib. Syst. Security*, vol. 20, 2012, p. 12.

[35] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Inference attack against encrypted range queries on outsourced databases," in *Proc. 4th ACM Conf. Data Appl. Security Privacy*, 2014, pp. 235–246.

[36] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic attacks on secure outsourced databases," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 1329–1340.

[37] K. G. Larsen and J. B. Nielsen, "Yes, there is an oblivious RAM lower bound!" in *Proc. Annu. Int. Cryptol. Conf.*, 2018, pp. 523–542.

[38] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "GhostRider: A hardware–software system for memory trace oblivious computation," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 87–101, 2015.

[39] C. Liu, M. Hicks, and E. Shi, "Memory trace oblivious program execution," in *Proc. IEEE 26th Comput. Security Found. Symp. (CSF)*, 2013, pp. 51–65.

[40] C. Liu, Y. Huang, E. Shi, J. Katz, and M. Hicks, "Automating efficient RAM-model secure computation," in *Proc. IEEE Symp. Security Privacy*, 2014, pp. 623–638.

[41] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: A programming framework for secure computation," in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 359–376.

[42] S. Lu and R. Ostrovsky, "Distributed oblivious RAM for secure two-party computation," in *Proc. Theory Cryptography*, 2013, pp. 377–396.

[43] M. Maas *et al.*, "PHANTOM: Practical oblivious computation in a secure processor," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 311–324.

[44] M. Maffei, G. Malavolta, M. Reinert, and D. Schröder, "Privacy and access control for outsourced personal records," in *Proc. IEEE Symp. Security Privacy (SP)*, 2015, pp. 341–358.
[45] T. Mayberry, E.-O. Blass, and A. H. Chan, "Efficient private file retrieval by combining ORAM and PIR," in *Proc. Netw. Distrib. Syst. Security*, 2014, pp. 1–11.
[46] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa, "Oblix: An efficient oblivious search index," in *Proc. IEEE Symp. Security Privacy*, 2018, pp. 279–296.
[47] M. Naveed, "The fallacy of composition of oblivious RAM and searchable encryption," IACR Cryptol. ePrint Archive, Rep. 2015/668, 2015.
[48] K. Nayak *et al.*, "HOP: Hardware makes obfuscation practical," in *Proc. Netw. Distrib. Syst. Security*, 2017, pp. 1–27.
[49] H. Omar, S. K. Haider, L. Ren, M. van Dijk, and O. Khan, "Breaking the oblivious RAM bandwidth wall," in *Proc. 36th IEEE Int. Conf. Comput. Design*, 2018, pp. 115–122.
[50] T. Onodera and T. Shibuya, "Succinct oblivious RAM," in *Proc. 35th Symp. Theor. Aspects Comput. Sci. (STACS)*, 2018, pp. 1–16.
[51] R. Ostrovsky and V. Shoup, "Private information storage," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 294–303.
[52] S. Patel, G. Persiano, M. Raykova, and K. Yeo, "PanORAMa: Oblivious RAM with logarithmic overhead," in *Proc. IEEE 59th Annu. Symp. Found. Comput. Sci. (FOCS)*, 2018, pp. 871–882.
[53] L. Ren *et al.*, "Constants count: Practical improvements to oblivious RAM," in *Proc. USENIX Security Symp.*, 2015, pp. 415–430.
[54] L. Ren, C. Fletcher, A. Kwon, M. van Dijk, and S. Devadas, "Design and implementation of the ascend secure processor," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 2, pp. 204–216, Mar./Apr. 2019.
[55] L. Ren, X. Yu, C. Fletcher, M. Van Dijk, and S. Devadas, "Design space exploration and optimization of path oblivious RAM in secure processors," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 571–582, 2013.
[56] C. Sahin, V. Zakhary, A. El Abbadi, H. Lin, and S. Tessaro, "TaoStore: Overcoming asynchronicity in oblivious data storage," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 198–217.
[57] S. Sasy, S. Gorbunov, and C. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," in *Proc. Netw. Distrib. Syst. Security*, 2018, pp. 1–15.
[58] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li, "Oblivious RAM with $O(\log^3 N)$ worst-case cost," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2011, pp. 197–214.
[59] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Security Privacy (S&P)*, 2000, pp. 44–55.
[60] E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," *J. ACM*, vol. 65, no. 4, p. 18, 2018.
[61] E. Stefanov and E. Shi, "Multi-cloud oblivious storage," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 247–258.
[62] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious RAM," in *Proc. Netw. Distrib. Syst. Security*, 2012, pp. 1–19.
[63] E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 299–310.
[64] R. Wang, Y. Zhang, and J. Yang, "Cooperative path-ORAM for effective memory bandwidth sharing in server settings," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2017, pp. 325–336.
[65] W. Wang *et al.*, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 2421–2434.
[66] X. Wang, H. Chan, and E. Shi, "Circuit ORAM: On tightness of the Goldreich–Ostrovsky lower bound," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 850–861.
[67] X. S. Wang, Y. Huang, T.-H. H. Chan, A. Shelat, and E. Shi, "SCORAM: Oblivious RAM for secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2014, pp. 191–202.
[68] X. S. Wang *et al.*, "Oblivious data structures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2014, pp. 215–226.
[69] P. Williams, R. Sion, and B. Carbunar, "Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, 2008, pp. 139–148.
[70] P. Williams, R. Sion, and A. Tomescu, "PrivateFS: A parallel oblivious file system," in *Proc. ACM Conf. Comput. Commun. Security (CCS)*, 2012, pp. 977–988.
[71] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 640–656.
[72] J. Yu, L. Hsiung, M. E. Hajj, and C. Fletcher, "Data oblivious ISA extensions for side channel-resistant and high performance computing," in *Proc. NDSS*, 2019, pp. 1–15.
[73] X. Yu *et al.*, "PrORAM: Dynamic prefetcher for oblivious RAM," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2015, pp. 616–628.
[74] X. Zhang *et al.*, "Shadow block: Accelerating ORAM accesses with data duplication," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2018, pp. 961–973.
[75] X. Zhang *et al.*, "Fork path: Improving efficiency of ORAM by removing redundant memory accesses," in *Proc. 48th Int. Symp. Microarchit.*, 2015, pp. 102–114.
[76] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. USENIX Security Symp.*, 2016, pp. 707–720.