[23] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC'87*, pp. 218–229. ACM Press, 1987.
[24] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM CCS'10*, pp. 451–462. ACM Press, 2010.
[25] W. Henecka and T. Schneider. Faster secure two-party computation with less memory. In *ASIACCS'13*, pp. 437–446. ACM Press, 2013.
[26] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ANSI C. In *ACM CCS'12*, pp. 772–783. ACM Press, 2012.
[27] Y. Huang, P. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *USENIX Conference on Hot Topics in Security (HotSec'13)*, pp. 4–4. USENIX Association, 2011.
[28] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security'11*, pp. 331–335. USENIX Association, 2011.
[29] F. Irigoin, P. Jouvelot, and R. Triolet. Semantical interprocedural parallelization: an overview of the PIPS project. In *International Conference on Supercomputing (ICS'91)*, pp. 244–251. ACM Press, 1991.
[30] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, pp. 145–161. Springer, 2003.
[31] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining'05*, pp. 593–599, 2005.
[32] F. Kerschbaum, T. Schneider, and A. Schröpfer. Automatic protocol selection in secure two-party computations. In *ACNS 14*, pp. 566–584. Springer, 2014.
[33] V. Kolesnikov and T. Schneider. Improved garbled circuit: free XOR gates and applications. In *ICALP'08*, pp. 486–498. Springer, 2008.
[34] B. Kreuter, A. Shelat, B. Mood, and K. R. B. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX Security'13*, pp. 321–336. USENIX Association, 2013.
[35] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security'12*, pp. 285–300. USENIX Association, 2012.
[36] Y. Lindell and A. Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In *ACM CCS'17*, pp. 259–276. ACM Press, 2017.
[37] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *IEEE S&P'15*, pp. 359–376. IEEE Computer Society Press, 2015.
[38] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM CCS'17*, pp. 619–631. ACM Press, 2017.
[39] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security'04*, pp. 287–302. USENIX Association, 2004.
[40] P. Mohassel and P. Rindal. ABY^3: a mixed protocol framework for machine learning. Cryptology ePrint Archive, Report 2018/403, 2018. `http://eprint.iacr.org/2018/403`.
[41] B. Mood, D. Gupta, H. Carter, K. R. B. Butler, and P. Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *IEEE EuroS&P'16*, pp. 112–127. IEEE, 2016.
[42] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce'99*, pp. 129–139. ACM, 1999.
[43] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P'13*, pp. 334–348. IEEE Computer Society Press, 2013.
[44] E. Pattuk, M. Kantarcioglu, H. Ulusoy, and B. Malin. CheapSMC: A framework to minimize secure multiparty computation cost in the cloud. In *DBSec'16*, pp. 285–294. Springer, 2016.
[45] A. Rastogi, M. A. Hammer, and M. Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *IEEE S&P'14*, pp. 655–670. IEEE Computer Society Press, 2014.
[46] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar. Chameleon: a hybrid secure computation framework for machine learning applications. In *ACM ASIACCS'18*. ACM Press, 2018.
[47] T. Schneider and M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC'13*, pp. 275–292. Springer, 2013.
[48] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: highly compressed and scalable sequential garbled circuits. In *IEEE S&P'15*, pp. 411–428. IEEE Computer Society Press, 2015.
[49] J. Vaidya and C. Clifton. Privacy-preserving *k*-means clustering over vertically partitioned data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining'13*, pp. 206–215. ACM Press, 2003.
[50] R. P. Wilson et al. SUIF: an infrastructure for research on parallelizing and optimizing compilers. *ACM SIGPLAN Notices*, 29(12):31–37, 1994.
[51] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *IEEE FOCS'86*, pp. 162–167. IEEE Computer Society Press, 1986.
[52] S. Zahur and D. Evans. Obliv-C: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153, 2015. `http://eprint.iacr.org/2015/1153`.
[53] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT'15*, pp. 220–250. Springer, 2015.
[54] Y. Zhang, A. Steele, and M. Blanton. PICCO: a general-purpose compiler for private distributed computation. In *ACM CCS'13*, pp. 813–826. ACM Press, 2013.

A FIXED-POINT COMPUTATIONS IN HYCC

In Listing 5 a code example is shown that implements 32 bit fixed-point numbers for ANSI C that can be used in any application compiled with HyCC.

```
#include <inttypes.h>

#define FP_BITS 32
#define FP_INTEGER_BITS 24
#define FP_FRACTION_BITS (FP_BITS - FP_INTEGER_BITS)

typedef int32_t fixedpt;
typedef int64_t fixedptd;

fixedpt fixedpt_mul(fixedpt a, fixedpt b)
{
  return  ((fixedptd)a * (fixedptd)b)  >>  \
             (fixedptd)FP_FRACTION_BITS);
}

fixedpt fixedpt_div(fixedpt a, fixedpt b)
{
  return ((fixedptd)a<<(fixedptd)FP_FRACTION_BITS)/b;
}
```

Listing 5: Code to add fixed-point support in ANSI C and thus, also applications compiled with HyCC.