Compacting Privacy-Preserving k-Nearest Neighbor Search using Logic Synthesis

Ebrahim M. Songhori
Dept. of ECE
Rice University
Houston, TX, USA
ebrahim@rice.edu

Siam U. Hussain
Dept. of ECE
Rice University
Houston, TX, USA
siam.umar@rice.edu

Ahmad-Reza Sadeghi
Technische Universität
Darmstadt, Germany
ahmad.sadeghi@
trust.cased.de

Farinaz Koushanfar
Dept. of ECE
Rice University
Houston, TX, USA
farinaz@rice.edu

ABSTRACT

This paper introduces the first efficient, scalable, and practical method for privacy-preserving k-nearest neighbors (k-NN) search. The approach enables performing the widely used k-NN search in sensitive scenarios where none of the parties reveal their information while they can still cooperatively find the nearest matches. The privacy preservation is based on the Yao's garbled circuit (GC) protocol. In contrast with the existing GC approaches that only accept function descriptions as combinational circuits, we suggest using sequential circuits. This work introduces novel transformations, such that the sequential description can be evaluated by interfacing with the existing GC schemes that only accept combinational circuits. We demonstrate a great efficiency in the memory required for realizing the secure k-NN search. The first-of-a-kind implementation of privacy preserving k-NN, utilizing the Synopsys Design Compiler on a conventional Intel processor demonstrates the applicability, efficiency, and scalability of the suggested methods.

Keywords

Secure Function Evaluation, Garbled Circuit, Logic Design, Logic Synthesis, Nearest Neighbor, k-NN, Data Mining, Privacy-Preserving

1. INTRODUCTION

The search for similarities has a wide range of applications in data mining, such as finding close matches in images, local features, biological and genome data, and multimedia systems [28]. The most extensively used function for similarity search is the k-nearest neighbor (k-NN). For a given dataset S of n points in a multi-dimensional space w, and a query q, the k-NN search finds a subset of S with k points that are closest to q. Numerous works have focused on development of efficient k-NN search, where the underlying assumption is that the dataset S and the query point q are public. For example, it is well known that the approximate search speed of k-NN can be dramatically improved by projecting the data into smaller hash tables [1, 31]. However, such approximation methods are orthogonal to our effort. We focus on the basic textbook k-NN search which could also be accelerated with the known approximations.

DAC '15, June 07-11, 2015, San Francisco, CA, USA
Copyright 2015 ACM 978-1-4503-3520-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2744769.2744808

In a number of scenarios, the data and the query are sensitive and it is important to maintain privacy while performing the search. This has motivated the development of privacy-preserving k-NN search [29]. The existing works assume that the parties each own a private dataset, while the query q is not private; they focus on devising higher-level protocols for privacy-preserving similarity search and proofs of privacy [28, 29]. These works mostly leverage homomorphic encryption to perform a *secure function evaluation (SFE)*. Homomorphic encryption enciphers the data (*plaintext*) in such a way that performing a mathematical function on the encrypted information, and then decrypting the result, produces the same answer as performing an analogous operation on the plaintext. Since the first fully homomorphic encryption was proposed about 5 years ago [8], numerous protocol-level and implementation-level advancements have been made. Even so, the implementations are still rather inefficient and impracticable for real applications.

This paper suggests the first efficient and scalable methodology for privacy-preserving k-NN search that is implementable on embedded processors. In contrast to the existing literature that assumes private datasets and known query, we assume a more general case of both private dataset and private query. Our methodology for providing a SFE is based on Yao's Garbled Circuit (GC) that is currently considered the most effective way to preserve privacy [4, 13]. Note that the use of GCs for the generic problem of privacy preserving data mining has been proposed, but not implemented [20]. GC requires that the function is represented as a binary circuit. It encrypts the truth tables of Boolean gates in the circuit. The input values are used as to decrypt the output value of the gates. All the garbled circuits suggested to date only support one pass, directed acyclic circuits, *a.k.a., combinational circuits.* The only available implementation of the privacy-preserving similarity search using the GC protocol is for the 1-NN search, where the circuit size was linearly increasing with the dataset size [17]. This increase is due to the fact that conventional combinational logic representation is not scalable.

To implement a GC, one needs to compile the higher-level description of the functions to the Boolean logic suitable for garbling. For this purpose, several custom compilers have been developed by the security and software/compiler communities [9, 11, 19, 23]. These compilers either use a custom library for a general purpose programming language [23], or introduce transformations for performing compile-time circuit garbling and on-the-fly gate generation [19]. Since these compilation methods are built upon the combinational logic model, they all suffer from the scalability issue.

Our work is the first to synthesize and optimize the GC for performing privacy preserving k-NN using a sequential representation. Instead of relying on custom compilers, we follow the recent TinyGarble methodology [30]. TinyGarble views the compilation of the general sequential circuit as a special case of logic synthesis. By defining new custom libraries and design objective/constraints, we demonstrate that utilizing standard logic synthesizers addresses the challenges in privacy preserving k-NN. As a result, we can store the GC and perform the privacy preserving k-NN search with an unprecedented efficiency.

Problem Statement. Alice has a query q and Bob has a dataset S. They want to jointly compute the k nearest neighbors of q in S such that Bob does not learn anything about q and Alice does not learn anything about S except the nearest neighbors.

Contributions. In brief, our contributions are as follows:

- Introducing the first efficient, practicable, and scalable methodology for privacy-preserving k-NN search assuming that the dataset and query are each privately held. The method is based on the Yao's GC protocol and implementable on embedded processors.
- Proposing a sequential circuit description for privacy-preserving k-NN search using Yao's Garbled Circuit protocol (instead of the known combinational representation). New transformations are created such that the sequential k-NN implementation are securely evaluated by interfacing with the available (combinational) cryptographic garbling schemes.
- Development of new custom libraries to generate optimized circuits for k-NN search using the standard logic synthesis tools. This is the first time that conventional logic synthesis is utilized for secure function evaluation/GC of k-NN.
- Reduction in the size of the required memory for GC from $\mathcal{O}(nw)$ to $\mathcal{O}(w)$ compared with the best known GC implementation of 1-NN [17]. Our scalable implementation requires a memory in the order of $\mathcal{O}(kw)$ for k-NN search. Note that k-NN search was impracticable earlier (for large n) due to the linear growth of the combinational representation.
- Proof-of-concept implementation of privacy preserving k-NN utilizing the Synopsys Design Compiler on an Intel processor. For example, the circuit size for k-NN search with $w = 31, k = 8$ is only 41.8KB.

2. PRELIMINARIES

Consistent with most literature in the area of GC and its implementations, we assume an honest but curious attack model [2, 9, 11, 19, 23].

2.1 Oblivious Transfer

Oblivious Transfer (OT) is a cryptographic protocol executed between a sender S and a receiver R, where R obliviously selects one of the inputs provided by S. More precisely, in an 1-out-of-n OT_1^n, S provides an n-tuple $(x_1, \ldots, x_n)$; R provides a selection number r with $1 \leq r \leq n$ and obtains x_r as the output [25]. A special case of an OT protocol is 1-out-of-2 for binary selection.

2.2 Yao's Garbled Circuits Protocol

Yao's garbled circuits [32] is a protocol between two parties, Alice and Bob, allowing them to compute a function on their secret inputs. More precisely, Alice, called creator, encrypts the function f to be computed where f is represented as a combinational Boolean circuit. To do this, the plain binary values are *garbled*-they are mapped to random symmetric *keys* and for each gate of the circuit, an encrypted truth table is generated that allows computation of gate's output key given its input keys. Alice sends the encrypted circuit along with her corresponding encrypted inputs to Bob, called evaluator. Now to evaluate f, Bob needs to obtain his inputs without revealing them to Alice. For this, Bob obtains his encrypted inputs obliviously through an 1-out-of-2 OT protocol and uses them to *evaluate* the encrypted function gate by gate. Finally, Alice provides a mapping from the encrypted output to plain output.

2.3 State-of-the-art GC optimizations

Our implementation adopts the state-of-the-art optimizations for garbled circuits, namely: free-XOR, row reduction, and garbling with a fixed-key cipher.

In the free-XOR method, Alice generates a global random $(k-1)$-bit value R which is known only to her. During garbling operation, she produces a key, X^0, for the binary value 0 and computes the other key X^1 as $X^1 = X^0 \oplus (R \parallel 1)^1$. With this convention, encrypting an XOR gate is simplified as it needs no cryptographic encryption and the keys for its output g can be computed as $X_g = X_a \oplus X_b$ where a and b are the inputs [18].

Using the row-reduction optimization reduces the size of the non-XOR gate truth tables by 25%. Instead of randomly emitting tokens for the gate output, they are generated as a function of the input tokens. Alice produces the output token such that the first entry of the garbled table becomes all 0. Thus, the first entry need not be sent [26].

Bellare et al. proposed garbling with a fixed-key cipher which results in an efficient garbling/evaluation of non-XOR gates by a fixed-key AES [2]. The proof of security given in their paper is in the random permutation model, as long as every AES call gets a unique-per-gate identifier. We adopt this approach as our GC scheme, while we devise new transformations to satisfy the uniqueness of the identifier.

3. RELATED WORK

The related literature in realizing privacy-preserving k-NN search has mainly focused on using homomorphic en-

[1] $\parallel$ denotes binary concatenation.

cryption as the enabling cryptographic primitive [28,29]. In their protocol, two parties perform k-NN search locally on their respective private dataset for a public query and then privately combine their results to form the k-NN. In contrast with these works, we adopt a more general setting in which one party holds a private dataset and the other one provides a private query. Moreover, we only rely on Yao's GC protocol which is known to be much more efficient SFE protocol than homomorphic encryption [4,13]. The use of GC for privacy preserving data mining has been suggested, but the existing literature focused on theoretical/protocol aspects and not implementation [20]. Leveraging our sequential description, this paper proposes the first scalable implementation and a low-overhead realization of secure k-NN on a conventional processor.

The work on generating Boolean functions for GC can be broadly classified into three categories: cryptographic primitives such as [2,23], transformations at the logic-level such as [18], and compiler/software techniques for mapping GC to the Boolean logic including [11,19,23]. Our work is orthogonal to the advances in the GC cryptographic primitives and logic-level transformations. We provide a compilation from the functional description to the Boolean logic which can be optimized and interfaced for any GC scheme. Therefore, we only describe the related work in the area of compiler/software techniques.

In the area of mapping and optimizing functions into Boolean logic for GC, the related literature has suggested developing custom compilers [9,11,19,23], custom libraries for conventional high-level compilers [10,14,22], hardware accelerators[2,16,27], and mobile device implementation [24].

Following the introduction of the first compiler for GC, called Fairplay [23], a number of researchers have focused on providing a custom compiler to interpret high level procedural language and map it to a circuit description language [9,11]. The most scalable existing compiler is PCF, which introduces loops that, if given manually in the high level language, are kept until the GC evaluation [19]. Our sequential description of the k-NN function that we input in the HDL format is much more compact than the high-level (software) loop embracing in PCF. Note that PCF is also based upon a combinational description.

Another class of proposed GC compilation methods leverage a library-based technique along with conventional software compilers. Some examples include FastGC [14], VMCRYPT [22], and FastGC extension to re-usable sub-circuits [10]. Our work is inspired by TinyGarble methodology [30]. We are the first to adapt a hardware description language and conventional logic synthesis for the important problem of privacy preserving of k-NN. Our method is automated, while it also benefits from custom logic-level libraries.

A number of researchers have suggested development of hardware accelerators for GC, including GPUs [15,27], FPGAs [16], or using the AES-NI available in recent CPUs [2]. Our work is orthogonal to this domain and can benefit from building accelerators for our sequential representation.

Using the GC for secure computing on resource constrained devices such as mobile/embedded platforms was suggested in [12]. A recent work in this area described a protocol for GC that relies on a smartcard embedded in the mobile device [7]. These implementations can greatly benefit from the scalable and efficient k-NN methodology introduced in this work. To overcome the limitations of resource-constrained devices, a set of relevant work in this area suggested outsourcing the GC generation and evaluation to cloud servers [5,6]. Our work demonstrates the feasibility of on-device GC implementation.

4. NEAREST NEIGHBORS SEARCH CIRCUIT

In this section, we present implementation of privacy-preserving k-NN search using Yao's GC protocol. First, we describe the optimized circuit generation for GC using logic synthesis tools. Next, we outline the implementation of 1-NN search using conventional GC based on combinational circuit. Lastly, we discuss the compact realizations of 1-NN and k-NN search based on sequential circuit.

Without any loss of generality, we assume the distance function for finding the nearest neighbors is Hamming distance in the following description.

4.1 Circuit Generation

We customize the flow of the standard logic synthesis tools to generate circuits optimized for GC protocol. As mentioned in Section 2.3 for GC with free-XOR optimization, there is a need to minimize the number of non-XOR gates in the Boolean representation. We perform two major customization in the synthesis flow. First, we create a new *synthesis library* to aid the conversion of the arithmetic and conditional operations to GC-optimized logical modules. Second, we develop a *technology library* to guide the mapping of the logic to the circuit netlist.

4.1.1 Synthesis Library

To realize the k-NN search, a set of basic arithmetic and conditional operations consisting of comparator, multiplexer, and Hamming distance are required. We create a custom synthesis library that includes the minimum non-XOR implementations of these operations. A w-bit comparator ($COMP_w$) is implemented using only w non-XOR gates [17]. A w-bit multiplexer (MUX_w) is realized using w non-XOR gates [18]. A w-bit Hamming distance ($HAMMING_w$) is devised using $w - \lceil \log_2(w) \rceil$ non-XOR gates where $w = 2^k - 1, k \in \mathbb{N}$ [3]. In all these modules, the total number of gates is $\mathcal{O}(w)$.

4.1.2 Technology Library

The technology library includes logical descriptions of basic units and their parameters like delay and area. The synthesis tool uses the technology library to generate a circuit optimized for given objectives and constraints. We design a custom technology library that contains 2-input gates (according to the requirement of the GC protocol). We set the area of XOR gates to 0 and the area of non-XOR gates to 1. The circuits are synthesized with the area constraint set to 0 so that the synthesis tool's objective becomes minimizing the number of non-XOR gates in the generated circuit.

An additional feature of this library is inclusion of non-standard gates (other than basic gates like NOT, AND, NAND, OR, NOR, XOR, and XNOR) to increase the flexibility of the mapping process. For example, the logical functions $F = A \vee B$ and $F = (\sim A) \vee B$ requires equal time in garbling/evaluation. However, using only standard gates the second function will require a NOT gate and an OR gate. We include four such non-standard gates which have

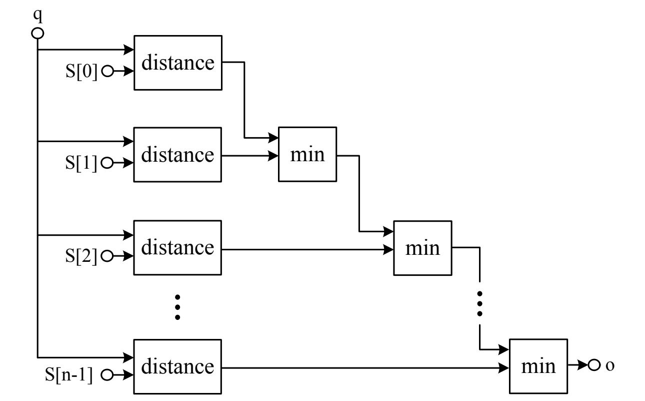

Figure 1: Combinational circuit for 1-NN. It consists of n Hamming distance and $n-1$ min modules.

an inverted input.

4.2 Combinational Garbled Circuit

As stated in Section 2.2, all previous implementations of GC protocol use a combinational description. To start our implementation for the special case of 1-NN search, we look for the closest point (o) to the query point (q) in the dataset (S). In the privacy-preserving setting, there is a need to compare the query point to all the points in the dataset. This is because the (private) intermediate search values cannot be utilized to bound the search, e.g., binary search.

Figure 1 shows the combinational circuit for 1-NN. The implementation uses n Hamming distance modules, and $n-1$ *min* modules (consisting of 1 COMP and 2 MUXs) to find the nearest point. One MUX selects the smaller distance for later comparison while the other one finds the point corresponding to that distance.

The total number of gates in the 1-NN combinational circuit is as follows:

$$\begin{aligned} \#\text{ of gates} = \; & n \times HAMMING_w \\ & + (n-1) \times (COMP_{\lceil \log_2(w) \rceil} \\ & + MUX_w + MUX_{\lceil \log_2(w) \rceil}) \\ \Rightarrow \#\text{ of gates} \in \; & \mathcal{O}(nw). \end{aligned}$$

The circuit should be garbled/evaluated only once. Thus, the time complexities of garbling/evaluation is $\mathcal{O}(nw)$.

4.3 Sequential Garbled Circuit

Sequential circuits can be used as a very compact circuit description for both real hardware and GC protocol. A sequential circuit is composed of a combinational circuit and a set of registers that stores the intermediate values. We modify the garbling scheme such that for each sequential cycle, it garbles/evaluates the combinational part and stores the garbling keys for the registers. The stored keys are used as inputs in the next cycle. To ensure security, each gate should have a unique identifier for each time that it is garbled/evaluated. Since in the sequential circuit each gate is garbled/evaluated multiple times, we use the combination of gate index and cycle index as a unique identifier for each gate invocation. Thereby, the proof of security provided in [2,21] also applies to our garbling scheme. We now describe the sequential 1-NN search implementation followed by k-NN implementation.

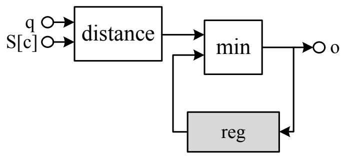

Figure 2: Sequential circuit for 1-NN. It consists of 1 Hamming distance and 1 min module. For a dataset of size n, the circuit is required to be garbled/evaluated n times.

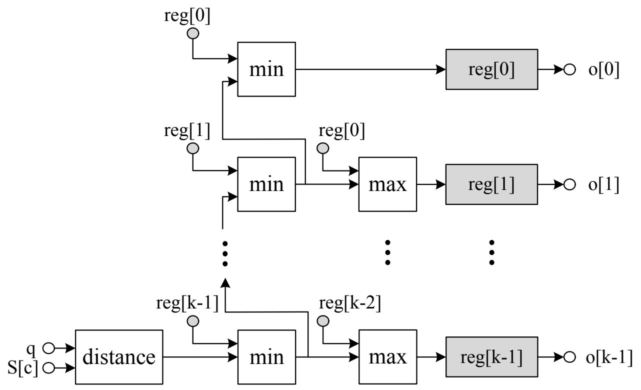

Figure 3: Sequential circuit for k-NN. It consists of 1 Hamming distance, k min, and $k-1$ max modules. It requires to be evaluated n times where n is the size of the dataset S.

4.3.1 Sequential 1-NN

Our 1-NN search sequential circuit is implemented with only 1 Hamming distance and 1 min module. Figure 2 illustrates the sequential circuit for 1-NN search. In each cycle c, the circuit computes the distance between q and $S[c]$. Next, it compares the resulting distance with the stored minimum distance in the register (reg). It then stores the minimum distance along with the nearest point until cycle c. The total number of cycles required to compute 1-NN is n.

The total number of gates in the 1-NN sequential circuit is as follows:

$$\begin{aligned} \#\text{ of gates} = \; & HAMMING_w \\ & + (COMP_{\lceil \log_2(w) \rceil} \\ & + MUX_w + MUX_{\lceil \log_2(w) \rceil}) \\ \Rightarrow \#\text{ of gates} \in \; & \mathcal{O}(w). \end{aligned}$$

The circuit should be garbled/evaluated n times. Thus, the time complexities of garbling/evaluation are the same as the combinational circuit and equal to $\mathcal{O}(nw)$.

4.3.2 Sequential k-NN

In k-NN search, the goal is to find the k nearest points to the query in the dataset. We expand the sequential circuit for the 1-NN to store the k nearest points. For this purpose, we implement a priority queue with depth of k which receives one point at each cycle. The priority of each point is equal to its distance to the query. Figure 3 shows the sequential circuit for the k-NN search. The circuit has 1 Hamming distance, k min, and $k-1$ *max* modules. The max module, like min, consists of 1 COMP and 2 MUXs.

The total number of gates in the 1-NN sequential circuit is as follows:

$$\begin{aligned}\#\text{ of nonXORs} &= \mathit{HAMMING}_w \\ &+ (2k-1) \times \mathit{COMP}_{\lceil \log_2(w) \rceil} \\ &+ (2k-1) \times (\mathit{MUX}_w + \mathit{MUX}_{\lceil \log_2(w) \rceil}) \\ \Rightarrow \#\text{ of nonXORs} &\in \mathcal{O}(kw).\end{aligned}$$

The circuit should be garbled/evaluated n times. Thus, the time complexities of garbling/evaluation are the same as the combinational circuit and equal to $\mathcal{O}(nkw)$. Note that due to the unscalability of combinational k-NN search, we did not include its implementation.

5. EVALUATION

The circuit generation is done with Synopsis Design Compiler (DC) 2010.03-SP4. We also use the Synopsis Library Compiler from DC package to interpret our custom technology libraries. For garbling and evaluation we use the JustGarble framework [2] which we modified to support sequential circuits. JustGarble exploits the cryptographic permutations realizable by fixed-key AES for GC operations. We run the framework on a system with Ubuntu 14.10 Desktop, 12GB of memory, and Intel Core i7-2600 CPU at 3.4GHz to assess the timing performance.

The metrics used to evaluate the performance of our implementations are as follows:

- The Circuit Size (CS) in Bytes is computed as

  $$CS = 24 \times q,$$

  where q is total number of gates. We store 2 indices (16B) and one type (8B) for each gate in circuit description file.
- Circuit Size Efficiency (CSE) is defined as

  $$CSE = \frac{CS_C}{CS_S},$$

  where CS_C is the size of the combinational circuit and CS_S is that of the sequential circuit.
- The garbling time is calculated as

  $$T = \#\text{ of non-XOR} \times T_{\text{non-XOR}} + \#\text{ of XOR} \times T_{\text{XOR}},$$

  where $T_{\text{non-XOR}}$ is the execution of a non-XOR gate and T_{XOR} is the execution of a XOR gate, which are 164 clock cycles (cc) and 62cc respectively in the specified system. We measured these values as the mean of 10,000 garbling trials.

The circuit size and timing evaluation for 1-NN search is reported in Table 1 for different values of w and n. For same w, the circuit size remains constant for sequential implementation. Therefore, the CSE increases linearly with n. As can be seen, the garbling times are almost equal for both combinational and sequential circuits. The small improvements in garbling time for sequential circuit is due to the more efficient optimization in the synthesis tool when circuit is small (sequential). For example where $n = 256$, $w = 31$, the time of garbling using combinational circuit is 6.83×10^6cc (2.01ms in our evaluation setup) while the one using sequential circuit is 6.24×10^6cc (1.84ms) which is 8.7% faster than combinational circuit.

The circuit size and timing evaluation for k-NN search is reported in Table 2. Since its combinational evaluation is not practical, we do not compare it in the way we did for 1-NN. The CS for the largest circuit in this work ($w = 31, k = 8$) is 41.8KB which will fit easily in embedded systems. As an example where $n = 128, w = 31, k = 8$, the garbling time would be $128k \times 173848\text{cc} = 22.8 \times 10^9\text{cc}$ (6.7s).

6. CONCLUSION

This paper presents an methodology for generation of highly compact and scalable privacy-preserving k-nearest neighbor (k-NN) search using garbled circuits (GC). We are the first to suggest a sequential description of k-NN, which enables generation of a compact Boolean GC. Our newly created custom libraries allow us to leverage the established powerful logic synthesis tools to optimize the (k-NN) Boolean expressions for GC. We introduce transformations that make the k-NN sequential expressions transparent to the available garbling schemes that are based on the combinational description. We show the first-of-a-kind implementation of privacy preserving k-NN using the Synopsys Design Compiler. It requires only 41.8KB storage for the 32-bit 8-NN search. Our implementation shows the practicability, efficiency, and scalability of the suggested methods.

7. ACKNOWLEDGEMENTS

We would like to thank anonymous reviewers for their helpful comments and suggestions to improve this work. This work is partially supported by an Office of Naval Research grant (ONR-R17460), a National Science Foundation grant (CNS-1059416), and U.S. Army Research Office grant (ARO-STIR-W911NF-14-1-0456) to ACES lab at Rice University. The work of the author at TU Darmstadt is in parts supported by the European Union's Seventh Framework Program (FP7/2007-2013) grant agreement n. 609611 (PRACTICE), the German Science Foundation (DFG) as part of project E3 within the CRC 1119 CROSSING, the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, and the Hessian LOEWE excellence initiative within CASED.

8. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, 2006.

[2] M. Bellare, V. Tung Hoang, Sriram K., and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *S&P*. IEEE, 2013.

[3] J. Boyar and R. Peralta. Concrete multiplicative complexity of symmetric functions. In *MFCS*. Springer, 2006.

[4] Brenner, perl, and Smith. hcrypt Secure Function Evaluation (SFE) project. `https://hcrypt.com/sfe/`.

[5] H. Carter, C. Lever, and P. Traynor. Whitewash: Outsourcing garbled circuit generation for mobile devices. In *ACSAC*. ACM, 2014.

[6] H. Carter, B. Mood, P. Traynor, and K. Butler. Secure outsourced garbled circuit evaluation for mobile phones. In *USENIX Security*. USENIX, 2013.

[7] D. Demmler, T. Schneider, and M. Zohner. Ad-hoc secure two-party computation on mobile devices using hardware tokens. In *USENIX Security*. USENIX, 2014.

[8] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

Table 1: Circuit size and timing evaluation for 1-NN search.

	W	7			15		31	
	N	128	256	512	128	256	128	256
Combinational	Total Gates	9044	18128	36304	19637	39349	40981	82069
	Non-XOR	2160	4336	8688	4325	8677	8530	17106
	CS(B)	217056	435072	871296	471288	944376	983544	1969656
	T(cc)	781048	1566208	3137024	1658644	3324692	3410882	6833090
Sequential	Total Gates	62			136		283	
	Non-XOR	17			34		67	
	CS(B)	1488			3264		6792	
	T(cc)	713984	1427968	2855936	1523200	3046400	3120640	6241280
Comparison	CSE	145.9	292.4	585.5	144.4	289.3	144.8	290.0

Table 2: Circuit size and timing evaluation for k-NN search.

W	7			15			31		
K	2	4	8	2	4	8	2	4	8
Total Gates	146	270	518	288	511	963	555	968	1784
Non-XOR	44	92	188	81	167	339	150	308	620
CS(B)	3504	6480	12432	6912	12264	23112	13320	23232	42816
T(cc)	13540	26124	51292	26118	48716	94284	49710	91432	173848

[9] W. Henecka, S. Kögl, A. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *CCS*. ACM, 2010.

[10] W. Henecka and T. Schneider. Faster secure two-party computation with less memory. In *ASIACCS*. ACM, 2013.

[11] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ANSI C. In *CCS*. ACM, 2012.

[12] Y. Huang, P. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *HotSec*. USENIX, 2011.

[13] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed Security Symposium (NDSS)*, 2012.

[14] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security*. USENIX, 2011.

[15] N. Husted, S. Myers, A. Shelat, and P. Grubbs. GPU and CPU parallelization of honest-but-curious secure two-party computation. In *ACSAC*. ACM, 2013.

[16] K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In *CHES*. Springer, 2010.

[17] V. Kolesnikov, A. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*. Springer, 2009.

[18] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *ICALP*. Springer, 2008.

[19] B. Kreuter, A. Shelat, B. Mood, and K. RB Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX Security*, 2013.

[20] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *CRYPTO*, 2000.

[21] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. *Journal of Cryptology*, 2009.

[22] L. Malka. Vmcrypt: modular software architecture for scalable secure computation. In *CCS*. ACM, 2011.

[23] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security*. ACM, 2004.

[24] B. Mood, L. Letaw, and K. Butler. Memory-efficient garbled circuit generation for mobile devices. In *FC*. Springer, 2012.

[25] M. Naor and B. Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 2005.

[26] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*. ACM, 1999.

[27] S. Pu and J. Liu. Computing privacy-preserving edit distance and Smith-Waterman problems on the GPU architecture, 2013.

[28] Y. Qi and M. J. Atallah. Efficient privacy-preserving k-nearest neighbor search. In *ICDCS*, 2008.

[29] M. Shaneck, Y. Kim, and V. Kumar. Privacy preserving nearest neighbor search. In *ICDMW*, 2006.

[30] E. M. Songhori, S. U. Hussain, A. Sadeghi, T. Schneider, and F. Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *S&P*. IEEE, 2015.

[31] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[32] Andrew C-C Yao. How to generate and exchange secrets. In *FOCS*. IEEE, 1986.