P3: Privacy Preserving Positioning for Smart Automotive Systems

by

Siam Umar Hussain

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

APPROVED, THESIS COMMITTEE:

Farinaz Koushanfar, Chair
Associate Professor of Electrical and Computer Engineering

Behnaam Aazhang
J.S. Abercrombie Professor of Electrical and Computer Engineering

Joseph R. Cavallaro
Professor of Electrical and Computer Engineering and Computer Science

Houston, Texas

December, 2015
P3: Privacy Preserving Positioning for Smart Automotive Systems

by

Siam Umar Hussain

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Farinaz Koushanfar, Chair
Associate Professor of Electrical and Computer Engineering

Behnaam Aazhang
J.S. Abercrombie Professor of Electrical and Computer Engineering

Joseph R. Cavallaro
Professor of Electrical and Computer Engineering and Computer Science

Houston, Texas

April, 2016
ABSTRACT

P3: Privacy Preserving Positioning for Smart Automotive Systems

by

Siam Umar Hussain

This thesis presents the first provably secure localization method for smart automotive systems. Using this method, a car, lost due to unavailability of GPS, can compute its location with assistance from three nearby cars while the locations of all the participating cars including the lost car remain private. Smart cars are increasingly being enhanced with smart features that boost both functionality and safety of the users. Two of the most substantial improvements in this field are smart navigation and inter-vehicle communications. However, in parallel to performance enhancement, these technologies have also created new dimensions for attacks. Therefore, security and privacy of the user should be taken into careful consideration while installing these advanced features. Previous approaches to maintain user location privacy suffered from one or more of the following drawbacks: trade-off between accuracy and privacy, one-sided privacy and need of a trusted third party that presents a single point to attack. The localization method presented here is one of the very first location-based services that eliminates all these drawbacks. The secure location is computed using a protocol utilizing the Secure Function Evaluation (SFE) technique named Yao’s Garbled Circuit (GC) that allows two parties to jointly evaluate a function on inputs which are encrypted to maintain privacy. The three assisting cars participate in a total six invocations of the 2-party GC operation to compute the
location of the lost car without revealing their location to one another. We design and optimize GC netlists for the functions required for computation of location by leveraging conventional logic synthesis tools with custom libraries optimized for GC. Proof-of-concept implementation of the protocol shows that the complete operation can be performed within only 550 ms. The fast computing time enables localization of even moving cars.
Acknowledgments

I owe my deepest gratitude to my advisor Prof. Farinaz Koushanfar. This thesis would not have been possible without her continuous support, encouragement, and guidance.

I am indebted to all my friends who have supported me throughout my life at Rice University. Especially, I would like to mention Ebrahim who have guided me since I started my graduate study at Rice.

Last but not the least, I express profound admiration and gratitude to my beloved parents and siblings for their endless love and unwavering support in my life.
# Contents

Abstract .......................................................... ii  
List of Illustrations ............................................. viii  
List of Tables .................................................... x  

1 Introduction ..................................................... 1  
1.1 Motivation .................................................... 1  
1.2 Approach ..................................................... 2  
1.3 Global Flow .................................................. 4  
1.4 Contributions ............................................... 5  

2 Related Work ................................................... 7  
2.1 Localization Methods ....................................... 7  
2.2 Privacy in Location Based Services ....................... 8  

3 Preliminaries ................................................... 11  
3.1 Cryptographic Protocols .................................... 11  
3.1.1 Threat Model ............................................. 11  
3.1.2 Oblivious Transfer ....................................... 12  
3.1.3 Secure Function Evaluation: Garbled Circuit .......... 12  
3.1.4 Generating Netlist with the TinyGarble Framework .. 14  
3.2 Triangle Localization ........................................ 17  

4 Protocol and Analysis .......................................... 19  
4.1 Protocol ...................................................... 19
Illustrations

1.1 Overview of the Localization Algorithm ................................................. 4

3.1 Triangle Localization Algorithm. The lost car is $Q$ and the assisting cars are $A$, $B$, and $C$. The calculated location of $Q$ is the centroid of the triangle $DEF$. ................................................................. 17

4.1 The regions of uncertainty for car $A$ in locating the other cars. The uncertainty region of the lost car $Q$ is marked with stripes and the uncertainty region of the other two assisting cars $B$ and $C$ is marked with dots. ................................................................. 24

5.1 Illustration of parallel invocations of GC protocol. .............................. 29

6.1 Illustration of the actual path, Euclidean distance and taxicab distance 33
6.2 Combinational circuit for 1-NN. It consists of $n$ taxicab distance and $(n - 1)$ min modules. ................................................................. 34
6.3 Sequential circuit for 1-NNS. It consists of 1 taxicab distance and 1 min module. For a dataset of size $n$, the circuit is required to be garbled/evaluated $n$ times. ................................................................. 36
6.4 Sequential circuit for $k$-NNS. It consists of 1 taxicab distance, $k$ min, and $k - 1$ max modules. It requires to be evaluated $n$ times where $n$ is the size of the dataset $S$. ................................................................. 37
7.1 Illustration of the error model ........................................... 38
7.2 Normalized mean error in the estimated location of the lost car as a function of the distance (normalized by $T$) between the actual location of the lost car and the median of the triangle formed by cars $A$, $B$, and $C$ with different degrees of distance compensation .... 39
7.3 Comparison of memory footprints of 1NNS with combinational and sequential approach ............................................. 41
7.4 Comparison of garbling times of 1NNS with combinational and sequential approach ............................................. 42
7.5 Memory footprint of $k$-NNS with sequential approach ............... 43
Tables

4.1 Inputs and outputs of the assisting cars at different steps of Phase 1 of the protocol .............................................. 21

5.1 Number of XOR and non-XOR for different functions of the synthesis library ..................................................... 28

7.1 Number of XOR and non-XOR gates in the netlists ...................... 40
7.2 Number of clock cycles required at different stages of garbling/evaluation of each netlist .............................................. 41
Chapter 1

Introduction

1.1 Motivation

Contemporary automobiles are increasingly being equipped with advanced technologies that make significant enhancements to both functionality and safety of the vehicles. Two of the most significant improvements in this field are smart navigation system and inter-vehicle communications. Each modern vehicle also includes an intra-network of processors connected to a central CPU providing Ethernet, USB, Bluetooth, and IEEE 802.11 interfaces [1]. Besides enhancing performance, these technologies create new dimensions for attack. Thus, in addition to classic vehicular reliability requirement, security and privacy of the user should be taken into careful consideration while implanting these advanced features [1–3]. Moreover, due to the increasing reliance on these smart features, backup plans to cope with the failure of one or more components is also crucial for reliability.

In this thesis, we present the first private localization method for smart cars based on provably secure primitives. With this method, a car, lost due to unavailability or malfunction of GPS, can send requests to three nearby cars to get assistance in finding its location. The three assisting cars then engage in a privacy preserving triangle localization protocol to estimate the location of the lost car. The locations of all the cars including the lost car, remain private.

One major use case for our privacy-preserving localization is in military applica-
tions when a lost military vehicle requires help in locating itself. It is crucial that the location of each participating vehicle remain private so that an adversarial vehicle cannot learn their locations by pretending to be an ally or by tapping into the common channel. This application can also be beneficial in verifying a suspected vehicles claimed location via distance bounding [1] with assistance from three nearby cars. Typically, three verifying base stations perform distance bounding on the suspected vehicle confining it to a triangular region. If the location reported by the vehicle itself falls within this triangle, the location is assumed to be accurate. However, this requires costly infrastructure which may not be available in all places. In this scenario, three other cars can act as the verifying base stations while their locations remain private.

1.2 Approach

Due to the emargance of mobile devices with positioning capabilty, location privacy has been an active area of research for about a decade. To date, the most widely explored method to ensure user privacy in Location Based Services (LBS) is location cloaking [4–6]. In this method, instead of sending the exact location and time instant of the user, a range of area covered in a period of time is sent. To make sure that user’s location cannot be inferred from this data, the range and period is chosen such that there are at least \( k - 1 \) other users in that area during that period, which ensures “\( k \)-anonymity” of the user. \( k \)-anonymity requires the existence of a trusted third party, called anonymizer that combines the user data with other users’ positions subscribed to the service. The anonymizer presents a single point to attack the system. Moreover, cloaking is also vulnerable to context based attack and trajectory-tracing. More importantly, the approximate location results in noisy and stochastic response
to the query. While this approximate response may be acceptable in some LBS scenario, for localization and navigation applications the accuracy of the method is crucial.

Earlier works in [7–9] explored performing the location-based query (e.g., nearest neighbor) in a transformed space. While these methods increase the accuracy over the cloaking approaches, they still have few drawbacks. For example, [7] proposes three methods that either requires a semi-trusted third party or has to sacrifice accuracy or privacy for simplified operation. The authors in [7, 9] consider the privacy of only one party (client), while the data of the other party (server) is assumed public.

To compute accurate results while maintaining complete privacy of all the participating parties, we employ Yao’s Garbled Circuit (GC) protocol [10] for Secure Function Evaluation which is currently considered to be the most effective provable privacy-preserving technique [11,12]. This protocol allows two parties to jointly evaluate a function on inputs which are encrypted to maintain privacy. Unlike the previous methods, this protocol does not involve trade-off between accuracy and privacy. To date, very few work have considered GC for LBS applications. Ours is the first privacy-preserving localization method based on GC.

We devise a protocol where the three assisting cars participate in a total six invocations of the 2-party GC operation such that the locations of all cars including the lost car remain private. To cope with the time constraint due to car movement, the protocol is designed such that each car can simultaneously participate in two GC operations with each of the two other cars (assuming a multi-core architecture of the processors, which is widely available at present).

In GC, the pertinent function is represented as a list of Boolean logic gates, called netlist. We generate the netlists required for the localization protocol by using conven-
tional logic synthesis tools with GC optimized custom libraries as suggested in [13]. Our custom synthesis library includes the first GC optimized implementations of division and square root functions, required for the computation of the location of the lost car. The synthesis library presented in [13] includes implementations of unsigned addition, subtraction, and multiplication. We added enhanced implementations of these functions to our library to support signed inputs and overflow.

![Figure 1.1: Overview of the Localization Algorithm](image)

**1.3 Global Flow**

The overview of the localization process is displayed in Fig. 1.1. The lost car $Q$ sends requests to three nearby cars $A$, $B$, and $C$ to assist in computing its location. Each assisting car $X (= A or B or C)$ estimates its distance $r_X$ from $Q$. Then $A$, $B$, and $C$ participate in a privacy preserving localization protocol based on Yao’s GC operation to compute the location of $Q$. The inputs from each assisting car $X$ are its location $L_X$ and its distance $r_X$ from $Q$. Ideally, the location of $Q$ would be a common intersection of three circles centered at $A$, $B$ and $C$. However, due to inaccuracy in distance estimation, the location of $Q$ is computed as the median of a triangle.
Each pair of cars (say $A$ and $B$) participates in a GC operation to compute two possible candidates for one vertex of the triangle. Then one of them (say $B$) participates in another GC operation with the third car ($C$) to select the candidate closer to $C$ as the vertex. Thus, six GC operations are required to determine all three vertices of the triangle. One car can learn zero to at most two vertices, therefore, a single car cannot compute the median on its own. Throughout the protocol, the input set $\{L_X, r_X\}$ of a car $X$ is not revealed to any other participating car. The median $L_Q$, the location of $Q$, is computed through secure sum [14] protocol where all four cars participate and revealed only to $Q$.

1.4 Contributions

In brief, our contributions are as follows:

- We present the first provably secure triangle localization for smart automotive systems. We design a protocol utilizing 2-party GC operation such that a lost car along with three nearby cars can jointly compute the location of the lost car while the locations of all the participating cars remain private.

- We develop a circuit synthesis library with functions required to generate GC optimized netlists for triangle localization algorithm. This library includes the first ever GC implementations of square-root and division operations.

- Proof-of-concept implementation of our protocol demonstrates practicality of the design. The complete protocol is performed within only 0.3 seconds which allows practical localization of cars even when they are moving.

In addition to the localization algorithm, this thesis presents a privacy preserving $k$-Nearest Neighbor Search ($k$-NNS). With recent surge in ride sharing services like
Uber, Lyft this $k$-NNS application will help maintain the privacy of both the riders and the drivers, while providing the intended service. In contrast with the existing GC approaches that only accept function descriptions as combinational circuits, we suggest using sequential circuits that results in a great efficiency in the memory required for realizing the secure $k$-NNS.

Recently the localization application has been accepted as a regular paper at the Design Automation Conference (DAC) 2016. An earlier version of the work on $k$-NNS has been published in DAC 2015.
Chapter 2

Related Work

In this chapter, we survey the literature related to our work. We first describe the currently available localization methods (generally non-privacy preserving) and explain our choice that is best suited for this work. Then we outline the techniques currently employed to ensure location privacy and explain their drawbacks.

2.1 Localization Methods

Till present localization algorithms have been mainly used in Wireless Sensor Networks (WSN) and in general does not consider the location of the participating parties as private data. These algorithms can be broadly divided into two categories: range-free and range-based. Range free localization techniques does not require distance estimation, instead, connectivity or proximity information from neighboring anchors are used. These techniques are used in applications where lower location accuracy is acceptable.

Range-based localization techniques on the other hand requires distance estimation. Different techniques are available in this category. In centroid localization, the unknown nodes location is set to the centroid of a polygon formed by the anchor nodes. The distance information is used to determine whether or not an anchor not should be a vertex of the polygon. Weighted centroid localization is an improved version of centroid localization where the centroid is calculated as the weighted mean
of the coordinates of the anchor nodes [15,16]. The weights are inversely proportional
to the estimated distance of the anchor nodes to the unknown node.

In triangle localization, three circles are drawn centered at three anchor nodes
with radius equal to the estimated distances from the unknown node [17–19]. A
triangle is formed with the intersections of the three pairs of circles as the vertices
and the location of the unknown node is set to the centroid of this triangle. In this
work we employ triangle localization as it require only three anchor nodes while for
the other techniques more anchor nodes are required for accuracy (In general many
anchor nodes are available in WSN, however for our application finding enough anchor
nodes, i.e., assisting cars is a challenge). We discuss triangle localization in details in
Section 3.2.

2.2 Privacy in Location Based Services

The works discussed above did not concern with the privacy of either the anchor nodes
or unknown nodes. The locations of all the nodes are assumed to be public. However,
there are some works that dealt with user privacy in Location Based Services (LBS).
As already mentioned, the most commonly used method has been location cloaking
[4–6] that have a few drawbacks. In this section, we discuss works that applied
cryptographic primitives in LBS scenarios.

There are a number of works that designed privacy preserving Location Based
Services (LBS) based on cryptographic primitives. Methods for privacy preserving
nearest neighbor search are presented in [7, 9]. The work in [7] employs one-way
Hilbert transformation to map the space of all elements to another space and resolve
the query in that transformed space. It requires a trusted third party to perform the
transformation in an offline phase. The method presented in [9] confines each point
of interest (POI) to a cell, named a Voronoi cell, such that the POI is the nearest neighbor to any point that falls within that cell. Then a regular rectangular grid is super-imposed over this Voronoi diagram. A user first decides which region on the grid she belongs to, then retrieves all the Voronoi cells intersecting that region through private information retrieval method and locally compute the nearest neighbor. Both these methods consider privacy of the query only, the database of the POIs is assumed to be public. Three methods based on homomorphic encryption to find if two friends are nearby without revealing their locations is presented in [8]. There are different trade-offs involved in these methods: they either require a semi-trusted third party or sacrifice accuracy or privacy for simplified operation.

The work in [20] presents application specific solutions based on GC to some problems in location-based services. They solve basic problems like point-inclusion (whether or not one party’s point is included in other party’s polygon), intersection (whether or not two polygons from two users have an intersection), closest pair (form a pair closest of points taking one point from each set provided by two users). A GC based method to compute the nearest neighbor of a group of people is presented in [21]. In this method, two users participates in GC protocol to compute the nearest neighbor of the group. The other members of that group receive their input keys through OT from the garbler and share them with the evaluator. We presented an scalable privacy preserving $k$-nearest neighbor search in [22]. It utilizes the sequential description of GC netlist introduced in [13]. This thesis includes an updated version of that work for use in location based services.

Note that, there are a number of work [23–26] on secure location verification using multilateration which focuses on ensuring that the suspect vehicle, called the prover, cannot provide a false location. However, none of these work take the privacy of
either the prover of verifier into consideration. Our work is related to the *point-in-triangle* test presented in [26] with the additional property of preserving privacy of the location of the participating parties. Designing a location verification scheme that is secure against false location claim while also preserves location privacy is part of our future work.
Chapter 3

Preliminaries

In this chapter we provide brief description of the background required to explain our work. We first explain the cryptographic tools we employ to ensure privacy while computing the location. Especially, we describe the garbling framework we use for secure computation and explain its superiority over previous frameworks. Finally, we describe the localization algorithm used to compute the location of the lost car.

3.1 Cryptographic Protocols

In this section, we provide a brief description of the cryptographic protocols employed in this secure localization. We first describe the threat model used in this work. Then we describe the Oblivious Transfer protocol that is employed in the Secure Function Evaluation (SFE) technique, Garble Circuit (GC) used in this work. Finally we explain the GC protocol and our GC framework TinyGarble.

3.1.1 Threat Model

Consistent with most work in this area, we assume an honest-but-curious attack model for our cryptographic protocols. In this model, the participating parties follow the agreed upon protocol, but may want to deduce more from the information at hand [27, 28]. This can be readily modified to support malicious model by following the methodologies presented in [29].
3.1.2 Oblivious Transfer

Oblivious Transfer (OT)[30] is a cryptographic protocol executed between a sender $S$ and a receiver $R$, where $R$ selects one from a pair of messages provided by $S$ without revealing her selection. In an 1-out-of-2 OT protocol, (OT$_2^1$), $S$ holds a pair of messages $(m_0, m_1)$; $R$ holds a selection bit $b \in 0, 1$ and obtains $m_b$ without revealing $b$ to $S$ and learns nothing about $m_{1-b}$.

3.1.3 Secure Function Evaluation: Garbled Circuit

Secure function evaluation (SFE) allows two or more parties to jointly compute a function of their respective private inputs. We employ the SFE technique called Yao’s Garbled Circuit (GC)[10]. In GC protocol, two parties Alice, called the Garbler and Bob, called the Evaluator, jointly compute a function $z = f(x, y)$ on their private inputs $x$, provided by Alice and $y$, provided by Bob. In the end, one or both of them learns the output $z$.

The steps of the GC protocol are as follows.

i. The function $f$ to be computed is represented as a Boolean circuit, called netlist, consisting of 2-input 1-output logic gates.

ii. Alice assigns each wire in the netlist with two $k$-bit random keys corresponding to the values 1 and 0.

iii. For each gate, she generates a garbled truth table by encrypting the keys for output with corresponding input keys.

iv. She then sends the garbled circuit along with the keys corresponding to her input values to Bob.
v. Bob obtains his keys corresponding to his input values obliviously through 1-out-of-2 OT protocol.

vi. He then uses these input keys to evaluate the encrypted tables gate by gate.

vii. Finally, Alice and Bob share their output maps, which can be configured to let one or both of them learn the output $z$.

**State-of-the-art GC optimizations**

A number of optimizations to the GC protocol have been proposed: free-XOR, row reduction, half gate, and fixed-key cipher. We provide a brief description of these optimizations here.

In the free-XOR optimization [31], Alice generates a random $(k - 1)$-bit private key $R$. During garbling, she generates a random $k$-bit key, $X^0$, for the value 0 of a wire and the key $X^1$, for the value 1 is generated as $X^1 = X^0 \oplus (R \| 1)$ *. With this convention, the keys for the output $g$ of an XOR gate with inputs $a$ and $b$ is computed as $X_g = X_a \oplus X_b$. Thus XOR gates do not need costly cryptographic encryption, which also translates to less communication time as the XOR gates does not need transfer of the garbled tables.

The size of the non-XOR gate truth table is reduced by 25% with the row-reduction optimization [32]. In this method, Alice produces the keys for the output of a gate as a function of the input keys such that the first entry of the garbled table becomes all 0 and need not be sent to Bob. The size of the non-XOR gate truth table is reduced by further 25% with half gate optimization presented in [33].

*\(\|\) denotes binary concatenation.
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 [28]. In this garbling scheme, the output key $X_g$ is encrypted with the input token $X_a$ and $X_b$ using the encryption function $E(X_a, X_b, T, X_g) = \pi(K) \oplus K \oplus X_g$, where $K = 2X_a \oplus 4X_b \oplus T$, $\pi$ is a fixed-key block cipher (e.g., instantiated with AES), and $T$ is a unique-per-gate number (e.g., gate identifier) called tweak. The proof of security is given in [28].

Our garbling framework TinyGarble [13], which we describe next, incorporates all these optimizations along with methods to generate netlists optimized for GC.

3.1.4 Generating Netlist with the TinyGarble Framework

Previous Approaches

Due to free-XOR optimization described in the previous section, the primary optimization goal while generating the netlist for the function $f$ is to minimize the number of non-XOR gates. The previous research on this phase of GC has followed two approaches: optimizations of cryptographic constructs and protocols such as [28, 31, 34–37], and compiler/engineering techniques such as [27, 38–43].

Two different techniques for circuit generation have been employed in the compiler/engineering realm. One technique is based on building a custom library for a general purpose programming language such as Java along with functions for creating the circuit, e.g., [40–42]. These libraries typically include frequently used modules such as adders and multipliers. However, library-based techniques require manual adjustment and do not perform global circuit optimization. Moreover, their memory management gets complicated when the number of gates is large thereby affecting performance and scalability [42].

The second technique is to design a new compiler for a higher-level language
that translates the instructions into the Boolean logic, e.g., [27, 38, 39, 43]. Although compiler-based techniques allow global optimizations, they often unroll the circuits into a large list of gates. For example, the description of a circuit with one billion gates has at least size $2 \log_2(10^9) \cdot 10^9 \approx 7$ GB. To reduce circuit description size, the compiler proposed in [27], called PCF (Portable Circuit Format), does not unroll the loops in the circuit until the GC protocol runs, and therefore seems to have a better scalability than the other compilers.

**TinyGarble Approach**

We follow the methodology presented in our garbling framework TinyGarble[13] to generate the netlist. TinyGarble simply views the circuit generation for GC as an atypical logic synthesis task that, if properly defined, can still be addressed by conventional hardware synthesis tools. This way, TinyGarble naturally benefits from the elegant algorithms and powerful techniques already incorporated in existing logic synthesis solutions.

**Synthesis Flow of TinyGarble**

In TinyGarble framework, $f$ is first described with a Hardware Description Language (HDL), like Verilog or VHDL, and compiled with a conventional logic synthesis tool to generate the GC netlist. There are two steps in this process. In the first step, the functional description of $f$ is converted into a structural representation consisting of standard logical elements. Then, this structural representation is converted into a netlist specific to the target platform. The two steps are described in details below.

**Synthesis Library.** The synthesis library is used to convert arithmetic operations like addition, multiplication, or more complex ones like square root and con-
ditional operations like if-else to their logical representations. The custom synthesis library of TinyGarble includes implementations of these function with the minimum number of non-XOR gates (in general synthesis tools try to minimize XOR gates as they require about four times more area than non-XOR gates, but in GC XOR gates are free). The arithmetic operations are based on a full adder with one non-XOR gate [44] and conditional operations are based on a 2-to-1 multiplexer (MUX) with one non-XOR gate [31].

**Technology Library.** The technology library is used to convert the logical representation to a netlist of Boolean gates. It includes logical descriptions of basic units and their parameters like delay and area. Along with standard Boolean logic (e.g., AND, OR, NAND etc) our technology library also includes non-standard logic (e.g., AND with an inverted input), which requires similar computation and communication in GC operation. To take advantage of the free XOR optimization the technology library is designed such that area of XOR, XNOR and NOT gates is 0 and the area of other non-XOR gates is 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 our custom technology library is that it contains non-standard gates (other than basic gates like NOT, AND, NAND, OR, NOR, XOR, and XNOR) to increase flexibility of mapping process. For example, the logical functions $F = A \land B$ and $F = (\neg A) \land B$ requires equal effort in garbling/evaluation. However by using only standard gates, the second function will require two gates (a NOT gate and an AND gate) and store one extra token for $\neg A$ in the memory. We include four such non-standard gates with an inverted input in our custom library.
3.2 Triangle Localization

Fig. 3.1 shows the setup of the triangle localization algorithm. The car $Q$ is lost. It requests three other cars $A$, $B$, and $C$ to help locate itself. First, distances $r_A$, $r_B$, and $r_C$ of $Q$ from $A$, $B$, and $C$ respectively, are estimated. In the ideal case where the estimated distance is exactly equal to the actual distance, the three circles centered at $A$, $B$, and $C$ with radii $r_A$, $r_B$, and $r_C$, respectively, would have a common intersection at $Q$. However, in practice distance cannot be estimated so precisely. An under estimation may result in no intersection. Therefore, the distance is generally overestimated. In this way, a triangle $DEF$ is formed by the points of intersections. The estimated location of $Q$ is the median of triangle.
In general, two circles intersects at two points (for example, circles withe centers at A and B intersect at F and F'). The one that falls inside the third circle forms one vertex of the triangle (F falls inside the circle centered at C). The equations for calculating the coordinates of F and F' is provided here [19]. The other intersections can be calculated in similar fashion.

\[
\sqrt{(x_F - x_A)^2 + (y_F - y_A)^2} = r_A \tag{3.1}
\]
\[
\sqrt{(x_F - x_B)^2 + (y_F - y_B)^2} = r_B \tag{3.2}
\]
\[
\sqrt{(x_F - x_C)^2 + (y_F - y_C)^2} \leq r_C \tag{3.3}
\]
\[
 x_F = \frac{1}{2p}(y_F q + t) \tag{3.4}
\]
\[
y_F = \frac{1}{p^2 + q^2}(pq x_A + y_B p^2 - \frac{1}{2}qt) \\
\pm \frac{1}{2} \sqrt{(qt - 2y_A p^2 - 2pq x_A)^2 - s(p^2 + q^2)} \tag{3.5}
\]

where,

\[
p = x_B - x_A, q = y_B - y_A
\]
\[
t = r_A^2 - r_B^2 + x_B - x_A + y_B - y_A
\]
\[
s = (4p^2 y_A^2 + t^2 - 4pt x_A + 4p^2 x_A^2 - 4p^2 r_A^2)
\]

Eq. (3.1) and (3.2) have two solutions as given by Eq.(3.4) and (3.5). The one that lies inside the range of C, decided through inequality (3.3), forms one vertex of the triangle. Note that, the vertex of the triangle is shown as F in the figure just for simplicity, it could be either of F or F'.
Chapter 4

Protocol and Analysis

The protocol for this privacy preserving localization involve two phases, and include six invocations of the two party GC protocol for two different functions. We first describe the protocol, then identify the best method for computing the distance between cars as required by the protocol and suggest some modification to the method to enhance privacy. Finally, we analyze the privacy it provides for the location of the participating cars.

4.1 Protocol

The protocol to securely compute the location of the lost car is described below. The lost car is denoted as $Q$ and the three assisting cars are denoted as $A$, $B$, and $C$. There are two phases of this protocol. In the first phase, the coordinates of the triangle $DEF$ are computed through six invocations of the GC protocol. For the location verification scenario, the coordinates are provided to the verifying authority after this phase. For other localization scenarios, the median of the triangle is computed through the Secure Sum[14] protocol in the second phase.

Phase 1: Computing triangle $DEF$

For this phase we need to evaluate the following two functions through GC. Similar to the previous section, the computation of the vertex $F$ is used as an example here.
\[ [x_F, y_F, x'_F, y'_F] = \text{Intersection}(x_A, y_A, r_A, x_B, y_B, r_B), \]
that implements Eq. (3.4) and (3.5).

\[ \text{in}_F = \text{Range}(x_F, y_F, x_C, y_C, r_C), \]
that implements inequality (3.3). The steps of this phase are as follows.

i. A, B, and C estimate their distances \( r_A, r_B, r_C \) respectively with \( Q \).

ii. A and B compute the coordinates \( F(x_F, y_F) \) and \( F'(x'_F, y'_F) \) of the intersections of their circles by evaluating the \text{Intersection} function though Yao’s GC protocol. The output map is configured such that A learns \( F(x_F, y_F) \) and B learns \( F'(x'_F, y'_F) \).

iii. B and C jointly decide whether \( F' \) lies inside the range of C by evaluating the \text{Range} function though Yao’s GC protocol. The output \( \text{in}_F \) is 1 if \( F' \) lies inside the range of C, and 0 otherwise, in which case the intersection \( F \) lies inside the range of C. B learns \( \text{in}_F \) and shares it with A. C learns nothing in this step.

iv. B and C perform the Step ii. algorithm. B learns \( D(x_D, y_D) \) and C learns \( D'(x'_D, y'_D) \).

v. C and A perform the Step iii. algorithm to compute \( \text{in}_D \) which is 1 if \( D' \) lies inside the range of A or 0 if D lies inside the range of A. C learns \( \text{in}_D \) and shares it with B. A learns nothing in this step.

vi. C and A perform the Step ii. algorithm. C learns \( E(x_E, y_E) \) and A learns \( E'(x'_A, y'_A) \).

vii. A and B perform the Step iii. algorithm to compute \( \text{in}_E \) which is 1 if \( E' \) lies inside the range of B or 0 if E lies inside the range of B. A learns \( \text{in}_E \) and shares it with C. B learns nothing in this step.
Table 4.1: Inputs and outputs of the assisting cars at different steps of Phase 1 of the protocol

<table>
<thead>
<tr>
<th>Step</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inputs</td>
<td>Outputs</td>
<td>Inputs</td>
</tr>
<tr>
<td>ii</td>
<td>$x_A, y_A, r_A$</td>
<td>$x_F, y_F$</td>
<td>$x_B, y_B, r_B$</td>
</tr>
<tr>
<td>iii</td>
<td>-</td>
<td>$in_F$</td>
<td>$x'_F, y'_F$</td>
</tr>
<tr>
<td>iv</td>
<td>-</td>
<td>-</td>
<td>$x_B, y_B, r_B$</td>
</tr>
<tr>
<td>v</td>
<td>$x_A, y_A, r_A$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>vi</td>
<td>$x_A, y_A, r_A$</td>
<td>$x'_E, y'_E$</td>
<td>-</td>
</tr>
<tr>
<td>vii</td>
<td>$x'_E, y'_E$</td>
<td>$in_E$</td>
<td>$x_B, y_B, r_B$</td>
</tr>
</tbody>
</table>

The inputs and outputs of the assisting cars at each step is shown in Table 4.1

**Phase 2: Computing the median of DEF**

After phase 1, each assisting car possesses the coordinates of two intersections and two Boolean variables indicating whether or not these intersections are vertices of DEF. In this phase, the assisting cars along with the lost car Q compute the median of the triangle through the following steps.

i  $Q$ sends a random coordinate $(x, y)$ to $A$.

ii $A$ computes the sums $X_A = (x + in_F.x_F + in_E.x'_E)$ and $Y_A = (y + in_F.y_F + in_E.y'_E)$ and sends to $B$.

iii $B$ computes the sums $X_B = (X_A + in_D.x_D + in_F.x'_F)$ and $Y_B = (Y_A + in_D.y_D + in_F.y'_F)$.
inF.y′F) and sends to C.

iv C computes the sums \( X_C = (X_B + \overline{m}_E.x_E + in_D.x'_D) \) and \( Y_C = (Y_B + \overline{m}_E.y_E + in_D.y'_D) \) and sends to Q.

v Q now subtracts the initial random numbers from the sums and compute the medians as \(((X_C - x)/3, (Y_C - y)/3)\) which are the coordinates of its location.

### 4.2 Distance Compensation

According to the protocol described in the previous section, one assisting car may know two vertices of DEF. The estimated location of Q is the median of the triangle DEF and is calculated through the secure sum protocol such that only Q learns the final location. However, if the area of the triangle is too small the location of Q may be estimated by a car with good accuracy from just two vertices of DEF. To prevent this, Q should be allowed to manipulate the area of DEF by controlling the estimated distances from the three assisting cars. On the other hand, the estimated distance should only be known to the respective assisting car.

Among several methods available for distance estimation like RSSI (Received Signal Strength Indicator)\cite{16, 17, 45}, TOA (Time of Arrival)\cite{45–47} , AOA (Angle of Arrival) \cite{48, 49} the one most suitable for this purpose is the two-way Time of Arrival method \cite{47}. In this method, the assisting car sends a synchronization message to the lost car and the lost car sends it back after some delay. Then, the assisting car measures the time shift (\( t_s \)) between the transmitted and received messages and subtract the estimated delay \( t_d \) to get the propagation time \( t_p = t_s - t_d \). In a typical application, the delay accounts for the time to receive the complete the message, and the time for the transceivers of both the cars to change their mode.
receiver). In this application, the lost car can wait an arbitrary time before sending back the message so that the actual delay is larger than the estimated delay $t_d$. This increases the estimated distance and eventually results in a larger area of $DEF$.

Note that since the final location is the median of the triangle, the larger area does not result in a significant error in the estimated location as we will show in Section 7.

4.3 Security Analysis

We now analyze what information the participating cars can learn regarding the location of the other cars through this protocol.

4.3.1 Lost Car

In the protocol described in this section, the lost car only participates in the computation in Phase 2 of the protocol and learns nothing but its own location. However, there is a maximum range within which the cars will be able to communicate with each other. If that range is $R$, the lost car can assume that the three assisting cars are within a circular area around it with a radius of $R$. Therefore the uncertainty over the location of the assisting cars is $1/\pi R^2$.

4.3.2 Assisting Cars

An assisting car can be interested in two types of information: the locations of the other two assisting cars and the location of the lost car. Each assisting car knows the coordinates of only one of the intersections with the circle of the other two assisting cars. Without the coordinates of the other intersection, it is not possible to deduce the center of the other circle. Therefore, the uncertainty for one assisting cars over the location of other two assisting cars is $1/\pi R^2$. 
Figure 4.1: The regions of uncertainty for car A in locating the other cars. The uncertainty region of the lost car Q is marked with stripes and the uncertainty region of the other two assisting cars B and C is marked with dots.

Regarding the location of the lost car, an assisting car knows the distance between the lost car and itself with some uncertainty created by the lost car by modifying the propagation time as described in Section 4.2. Therefore, an assisting car X (= A or B or C) can confine the location of the lost car within a circular region with radius $r_X$. It is possible for one assisting car to know the coordinates of two of the vertices of the triangle DEF. Those two vertices form one chord of that circle. In a strict sense, it is not possible to learn which side of that chord the other vertex resides. However, if the two partitions on either side of the chord have largely different areas, it is more likely that the other vertex is on the larger partition. Even though it is not straightforward to calculate the uncertainty here, the minimum uncertainty in this
case would be $2/\pi r_x^2$.

The regions of uncertainty for car $A$ in locating the other cars is shown in Fig. 4.1. The uncertainty region of the lost car $Q$ is marked with stripes and the uncertainty region of the other two assisting cars $B$ and $C$ is marked with dots. It is assumed that $A$ knows the vertices $E$ and $F$ of the triangle $DEF$. 
Chapter 5

GC Operation

As explained in Section 3.1.3 we need to generate a netlist consisting of Boolean logic, with the optimization goal set to minimizing the number of non-XOR gates, to securely evaluate a function. In this chapter, we describe the generation of these netlists with our custom synthesis library and the invocation of the GC protocol to securely evaluate these netlists.

5.1 Netlist Generation

We follow the TinyGarble methodology[13] to generate the netlists for the intersection and range functions. Even though TinyGarble supports both sequential and combinational circuits, the later approach is more suited for the localization application as it does not involve repeated operation for most of the parts. The TinyGarble framework provides free-XOR optimized synthesis library that contains implementations of arithmetic functions like unsigned addition, subtraction, and multiplication. For implementations of (3.3) - (3.5) we extend the library by including signed versions of these functions along with support for overflow. In addition to this, we implemented free-XOR optimized division and square-root functions as required by (3.4) and (3.5).
5.1.1 Intersection

To generate the GC optimized netlist for the Intersection function that computes Eq. (3.4) and (3.5) we need the implementations of arithmetic functions with the minimum number of non-XOR gates, which minimizes both the number of communication and computation [31]. We follow the TinyGarble methodology [13] with custom synthesis and technology libraries to generate the netlists. Our custom synthesis library includes the first GC optimized implementations of the division and square-root functions. Moreover, implementations of all the arithmetic functions support signed inputs with variable bit-length and overflow, which are essential for generating netlist for an arbitrary practical function.

In our implementation, complexity of the number of non-XOR gates in a \( W \) bit division operation is \( O(W^2) \) which is similar to the complexity of the multiplication operation provided in [13]. The number of non-XOR gates for a 64-bit division operation is 12,546.

The square root operation follows an iterative procedure. Complexity of the number of non-XOR gates in a \( W \) bit square root operation with \( K \) iterations is \( O(W^2K) \). Again, the number of required iterations can be assumed to be linearly proportional to the bit width, which simplifies the term to \( O(W^3) \). The number of non-XOR gates for a 64-bit square root operation with 32 iterations is 12,733.

5.1.2 Range

Even though inequality (3.3) involves square-root operation, both sides of this inequality are positive quantities as both of them represent distances. Therefore, we can avoid the costly square-root operation by squaring both sides. As a result, the netlist for this function is much smaller than the Intersection netlist.
The netlists for each function need to be generated only once. It is generated offline and saved in each car’s memory. The number of non-XOR and XOR gates for different functions of the synthesis library is presented in Table 5.1.

Table 5.1 : Number of XOR and non-XOR for different functions of the synthesis library

<table>
<thead>
<tr>
<th>Function</th>
<th>Bit Length</th>
<th>Number of gates</th>
<th>Non-XOR</th>
<th>XOR</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addition</td>
<td>16</td>
<td></td>
<td>16</td>
<td>47</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>32</td>
<td>95</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>64</td>
<td>191</td>
<td>255</td>
</tr>
<tr>
<td>Subtraction</td>
<td>16</td>
<td></td>
<td>16</td>
<td>36</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>32</td>
<td>68</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>64</td>
<td>132</td>
<td>196</td>
</tr>
<tr>
<td>Comparison</td>
<td>16</td>
<td></td>
<td>16</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>32</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>64</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Multiplication</td>
<td>16</td>
<td></td>
<td>618</td>
<td>966</td>
<td>1,584</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>2,263</td>
<td>3,454</td>
<td>5,717</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>8,637</td>
<td>10,425</td>
<td>19,062</td>
</tr>
<tr>
<td>Division</td>
<td>16</td>
<td></td>
<td>830</td>
<td>961</td>
<td>1,791</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>3,185</td>
<td>4,315</td>
<td>7,500</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>12,546</td>
<td>14,158</td>
<td>26,704</td>
</tr>
<tr>
<td>Square Root</td>
<td>16</td>
<td></td>
<td>294</td>
<td>451</td>
<td>745</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>2,013</td>
<td>5,111</td>
<td>7,124</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>12,733</td>
<td>51,579</td>
<td>64,312</td>
</tr>
</tbody>
</table>
5.2 Invocation of GC Protocol

Each of the assisting cars participates in two GC operations on the Intersection function with the other two cars. These two GC operations are independent of each other and performed in parallel in two cores of the processor. To ensure symmetry, each car performs as the garbler for one pair and the evaluator for the other. Similarly, each assisting car participates in two parallel GC operations on the Range function with the other two cars. Fig. 5.1 illustrates these operations. The outer arrows depict GC on Intersection and the inner arrows depict GC on Range. The vertex of DEF that is being computed in each GC operation is also indicated beside the arrows. A solid arrow emanating from a car indicates that the car acts as the garbler in that operation, and a dashed arrow indicates the evaluator.

The operation of the car A is described here as an example. A acts as the garbler while B acts as the evaluator to determine the coordinates of F and F’ through the Intersection function and only learns the coordinate of F. In parallel to this,
A participates in another GC operation as the evaluator, with C as the garbler to compute the coordinates of E and E’ and learns only the coordinate of E’. A then performs as the garbler, while B performs as the evaluator to decide whether E’ forms one vertex of the triangle through the Range function and shares the result with C. At the same time, it acts as the evaluator in another GC operation where C is the garbler to decide whether D’ forms one vertex of the triangle without learning the result.
Chapter 6

Privacy Preserving k-Nearest Neighbor Search*

Recently ride sharing apps like Uber, Lyft have become popular both as cheap rides and a source of income by providing rides. With the emerge of these services concern over the privacy of both the riders and the drivers have gained attention. To ensure location privacy of both the parties, we need to design a system that allows the rider to search for the nearest car without revealing her location to the service provider while ensuring that the rider only knows about the few drivers who are nearby.

We present the first efficient and scalable methodology for privacy-preserving $k$-NN search that is implementable on embedded processors. We have already described other approaches to perform privacy preserving $k$-NNS and discussed their limitations. Beside those works, 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 [50]. This increase is due to the fact that conventional combinational logic representation that was employed in that implementation is not scalable.

We present the first efficient, practicable, and scalable methodology for privacy-preserving $k$-NN search based on yao’s GC protocol. It utilizes the sequential circuit description for GC instead of the conventional combinational representation. It also benefits from the custom libraries presented in TinyGarble [13], as already discussed.

*This work was done in collaboration with Ebrahim Songhori
As a result, we can store the GC and perform the privacy preserving $k$-NN search with an unprecedented efficiency.

**Problem Statement.** The rider, Alice has a query $q$, which is her location and the service provider, Bob has a dataset $S$ containing the location of the available drivers in that area. 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 drivers.

Our work reduces the size of the required memory for GC from $O(nw)$ to $O(w)$ compared with the best known GC implementation of 1-NN [50]. Our scalable implementation requires a memory in the order of $\mathcal{O}(kw)$ for $k$-NNS search. Note that $k$-NN search was impracticable earlier (for $k > 1$) 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 with $w = 31, k = 8$ requires only 80KB of memory.

### 6.1 Distance Function

For $k$-NNS in 2D space, the default distance function would be Euclidean distance, which computes the length of the straight line path between the two points. However, in practice there is almost never a straight line path between two cars on the road as demonstrated with an example in Figure 6.1. We employ a more practical and computationally efficient taxicab distance. The taxicab distance, $d_t$ between two points with rectangular coordinates $(x_1, y_1)$ and $(x_2, y_2)$ is given by

$$d_t = (|x_1 - x_2| + |y_1 - y_2|)$$
As evident from the example in Figure 6.1, this distance function closely resembles the actual distance the driver has to cover to reach the rider.

Note that, the $k$-NNS presented here is compatible with any distance function. For example, in the generic $k$-NNS presented in [22], the distance function was Hamming distance.

### 6.2 Generation of Netlist

As already mentioned, all the circuits are synthesized using the methodology presented in Section 3.1.4 To realize the $k$-NNS, a set of basic arithmetic and conditional operations consisting of comparator, multiplexer, and distance function 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 [50]. A $w$-bit multiplexer ($MUX_w$) is realized using $w$ non-XOR gates [31]. A $w$-bit taxicab distance ($TD_w$) is devised using $7w + 1$ non-XOR gates. In all these modules, the total number of gates is $\mathcal{O}(w)$.
Figure 6.2: Combinational circuit for 1-NN. It consists of \( n \) taxicab distance and \((n-1)\) min modules.

### 6.3 Combinational Garbled Circuit

All previous implementations of GC protocol use a combinational description. To start our implementation for the special case of 1-NNS, 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 6.2 shows the combinational circuit for 1-NNS. The implementation uses \( n \) taxicab distance modules, and \((n-1)\) \textit{min} modules (consisting of 1 \text{COMP} and 2 \text{MUX}s) 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-NNS combinational circuit is as follows:

\[
\text{\# of gates} = n \times TD_w + (n - 1) \times (COMP_{w+1} + 2MUX_{w+1})
\]

\[
\Rightarrow \text{\# of gates} \in O(nw).
\]

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

6.4 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 [28,51] also applies to our garbling scheme. We now describe the sequential 1-NNS implementation followed by \(k\)-NNS implementation.

6.4.1 Sequential 1-NNS

Our 1-NNS sequential circuit is implemented with only 1 taxicab distance and 1 min module. Figure 6.3 illustrates the sequential circuit for 1-NNS. In each cycle \(c\), the circuit computes the distance between \(q\) and \(S[c]\). Next, it compares the resulting
Figure 6.3: Sequential circuit for 1-NNS. It consists of 1 taxicab distance and 1 min module. For a dataset of size $n$, the circuit is required to be garbled/evaluated $n$ times.

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-NNS is $n$.

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

$$\text{# of gates} = TD_w + COMP_{w+1} + 2MUX_{w+1}$$

$$\Rightarrow \text{# of gates} \in \mathcal{O}(w).$$

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 $\mathcal{O}(nw)$.

6.4.2 Sequential $k$-NNS

In $k$-NNS, the goal is to find the $k$ nearest points to the query in the dataset. We expand the sequential circuit for the 1-NNS 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 6.4 shows the sequential circuit for the $k$-NNS. The circuit has 1 taxicab 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-NNS sequential circuit is as follows:

\[
\# \text{ of nonXORs} = TD_w + (2k - 1) \times COMP_{w+1} + 2(2k - 1) \times MUX_{w+1}
\]

\[\Rightarrow \# \text{ of nonXORs} \in \mathcal{O}(kw).\]

The circuit should be garbled/evaluated \(n\) times. Thus, the time complexity of garbling/evaluation is equal to \(\mathcal{O}(nk\text{w})\). Note that due to the unscalability of combinational \(k\)-NNS, we did not include its implementation.
Chapter 7

Evaluation

In this chapter, we first analyze the error in location measurement associated with the triangle localization algorithm. Next, we evaluate the two netlists required for the secure localization protocol in terms of the number of non-XOR gates. Then, we garble/evaluate them through the GC framework and present the timing results. Finally we evaluate the nearest neighbor implementations in terms of memory footprint and timing.

Figure 7.1: Illustration of the error model
7.1 Error Analysis of Localization Algorithm

We first analyze the error in the location estimated by the triangle localization algorithm. Note that this error is solely due to the localization method, and distance estimation error. The protocol does not introduce any additional error.

To estimate the error, we run simulation by placing the assisting cars at random positions inside a square area with dimension $T$ and place the lost car at the center of that square. The error is quantified as the Euclidean distance between the estimated and actual location of the lost car, normalized to $T$. Since the estimation error depends on the positions of the assisting cars with respect to the lost car, the error is plotted against the distance (normalized to $T$) between the actual location of the lost car and the median of the triangle formed by cars $A$, $B$, and $C$ as illustrated in Fig. 7.1.

![Figure 7.1](image)

**Figure 7.1**: Normalized mean error in the estimated location of the lost car as a function of the distance (normalized by $T$) between the actual location of the lost car and the median of the triangle formed by cars $A$, $B$, and $C$ with different degrees of distance compensation.

Fig. 7.2 shows the result. For each point on the curves, the simulation is run for
5.7E + 03 times. To analyze the effect of distance compensation, we simulate three cases where the actual distance is increased by 50%, 70%, and 90%, respectively. The plot shows that the estimation errors are fairly close for all three cases.

7.2 Circuit Synthesis for Localization Protocol

Two netlists are required for the GC operations- Intersection and Range. The equations for these functions are described using Verilog HDL and compiled with Synopsys Design Compiler [52] with our custom libraries. The number of non-XOR and XOR gates in the netlists are presented in Table 7.1. As already mentioned, the garbling/evaluation time depends only on the number of non-XOR gates.

Table 7.1 : Number of XOR and non-XOR gates in the netlists

<table>
<thead>
<tr>
<th>Function</th>
<th>Intersection</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of non-XOR gates</td>
<td>5.38E+04</td>
<td>7.83E+02</td>
</tr>
<tr>
<td>No. of XOR gates</td>
<td>1.74E+05</td>
<td>1.71E+03</td>
</tr>
<tr>
<td>Total</td>
<td>2.27E+05</td>
<td>2.49E+03</td>
</tr>
</tbody>
</table>

7.3 Timing of Localization Protocol

To assess the timing performance, we run the localization protocol on a system with Ubuntu 14.10 Desktop, 12.0 GB of memory, and Intel Core i7-2600 CPU @ 3.4GHz using TinyGarble framework. The number of clock cycles required at different stages of garbling/evaluation of each netlist once is presented in Table 7.2.

Each netlist is garbled/evaluated 3 times by the three cars in parallel. The total number of clock cycles from the lost car initiating the operation to the final compu-
Table 7.2: Number of clock cycles required at different stages of garbling/evaluation of each netlist

<table>
<thead>
<tr>
<th>Function</th>
<th>Intersection</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Garbler</td>
<td>Evaluator</td>
</tr>
<tr>
<td>OT</td>
<td>4.97E+08</td>
<td>5.54E+08</td>
</tr>
<tr>
<td>Communication</td>
<td>1.19E+06</td>
<td>3.42E+07</td>
</tr>
<tr>
<td>Garbling/Evaluation</td>
<td>3.28E+07</td>
<td>2.11E+07</td>
</tr>
</tbody>
</table>

tation of the location is $1.89E+09$ which translates to only 550 ms on our current implementation platform.

Figure 7.3: Comparison of memory footprints of 1NNS with combinational and sequential approach

### 7.4 Memory Footprint of 1-NNS

We compare our approach with previous ones for the special case of $k = 1$ since higher values of $k$ were prohibitive with the previous approach. The memory footprint (circuit size) depends on the total number of gates in the circuit. Figure 7.3 shows the
total number of gates as a function of the input word length, \( w \) and library size, \( n \). We observe that while the memory footprint increases linearly with \( n \) for combinational approach, it is independent of \( n \) for sequential approach. Moreover, the circuit size is orders of magnitude smaller with our approach.

### 7.5 Timing of 1-NNS

The garbling/evaluation time is proportional to the total number of non-XOR gates that needs to be garbled. Theoretically, garbling time for both combinational and sequential approach should be similar. However, as shown in Figure 7.4 the computation time is reduced with sequential approach. This has two reasons. First, with reduction in circuit size, optimization by the logic synthesis tools is more effective resulting in reduction in the number of non-XOR gates. Second, with lower memory footprint for sequential circuit, there are fewer cache misses resulting in faster operation. The results are obtained with the setup described in Section 7.3.

![Figure 7.4: Comparison of garbling times of 1NNS with combinational and sequential approach](image)

Figure 7.4: Comparison of garbling times of 1NNS with combinational and sequential approach
7.6 Memory Footprint of $k$-NNS

Figure 7.5 shows the total number of gates in sequential $k$-NNS circuit as a function of the input word length, $W$ and $k$. As expected, it increases linearly with both $W$ and $k$. As already explained, the total number of gates is independent of the library size, $N$.

Figure 7.5: Memory footprint of $k$-NNS with sequential approach

The actual memory footprint for the largest circuit in this work ($w = 31, k = 8$) is 80KB which will fit easily in an embedded systems.
Chapter 8

Conclusion

We present the first provably secure localization method for smart vehicles. We devise a protocol that allows a lost car compute its location with assistance from three nearby cars through privacy-preserving computation such that locations of all the cars remain private. The protocol employs the well known SFE technique named Yao’s Garbled Circuit (GC) for the computations jointly performed by the cars to determine the location of the lost car without revealing their own locations to any other car.

In addition, we present a methodology for generation of highly compact and scalable privacy-preserving $k$-nearest neighbor search ($k$-NNS) using garbled circuits (GC). We are the first to suggest a sequential description of $k$-NNS, which enables generation of a compact Boolean GC.

These applications are two of the very first location-based services that does not involve any trade-off between accuracy and privacy. We design netlists for the functions required for computation of location and compiled them with conventional logic synthesis tool using custom libraries that incorporate implementations of arithmetic operations optimized for the GC protocol. Our implementation demonstrates that the localization operation is completed within only 550 ms, a time period short enough to localize moving cars.
8.1 Future Work

In the current work, we assume that the lost car is honest about its location. In the location verification scenario, the lost car, in this case called “the prover” may want to report false location. As already mentioned, a number of work [23–26] focus on preventing false location claims. However, none of these work take the privacy of either the prover or verifier into consideration. Our next goal is to design a privacy preserving localization scheme that is also resilient against false location claim.

Our current implementation follows a combinational approach as it the sequential operation would incur significant overhead in the current version of TinyGarble. The next version will reduce the sequential overhead close to zero allowing sequential implementation of the localization protocol.

Moreover, since the secure localization protocol is designed for localization of cars, our next goal is to implement the protocol in platforms specialized for automotive applications. Current implementation of our GC framework TinyGarble utilizes AES-NI instruction set for encryption required in GC. This instruction set is available only in select Intel processors. We plan to implement the protocol on Intel NUC Kit with Intel Atom Processor E3815[53], which supports AES-NI. Another potential platform is the MPC5777M Micro-controller[54], which also targets automotive applications. For AES in MPC5777M, our implementation will utilize AES-128 block on the Hardware Security Module.
Bibliography


