Securely Computing a Ground Speed Model

EYAL KOLMAN, RSA
BENNY PINKAS, Bar Ilan University

Consider a server offering risk assessment services and potential clients of these services. The risk assessment model that is run by the server is based on current and historical data of the clients. However, the clients might prefer not sharing such sensitive data with external parties such as the server, and the server might consider the possession of this data as a liability rather than an asset. Secure multi-party computation (MPC) enables one, in principle, to compute any function while hiding the inputs to the function, and would thus enable the computation of the risk assessment model while hiding the client's data from the server. However, a direct application of a generic MPC solution to this problem is rather inefficient due to the large scale of the data and the complexity of the function.

We examine a specific case of risk assessment—the ground speed model. In this model, the geographical locations of successive user-authentication attempts are compared, and a warning flag is raised if the physical speed required to move between these locations is greater than some threshold, and some other conditions, such as authentication from two related networks, do not hold. We describe a very efficient secure computation solution that is tailored for this problem. This solution demonstrates that a risk model can be applied over encrypted data with sufficient efficiency to fit the requirements of commercial systems.

CCS Concepts: • **Security and privacy** → **Security services**; **Privacy-preserving protocols**

Additional Key Words and Phrases: Secure multi-party computation, risk models

ACM Reference Format:
Eyal Kolman and Benny Pinkas. 2017. Securely computing a ground speed model. ACM Trans. Intell. Syst. Technol. 8, 4, Article 54 (June 2017), 13 pages.
DOI: http://dx.doi.org/10.1145/2998550

1. INTRODUCTION

One of the main threats in IT security is impersonation, where an attacker steals the credentials of a legitimate user and penetrates into a secure network. Once inside the network, the attacker is able to delete, modify, or steal data. Hence, impersonation detection and user authentication are a crucial part of security systems.

Advanced impersonation detection systems quantify the risk that someone is impersonating a user who has rights to access a resource. An example of an impersonation detection system is a web server of an online bank that receives login attempts from people wishing to access accounts. In response to a login attempt to access a user's account, the web server extracts any of a number of features describing the login attempt, such as the user's IP address, geolocation, login time, and the like. The web server then quantifies the risk that the login was attempted by an imposter by inputting the

This work was supported by a Magneton grant of the Israeli Ministry of Economy, and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minster's Office.
Authors' addresses: E. Kolman, RSA Security, Herzliya, Israel; email: eyal.kolman@rsa.com; B. Pinkas, Dept. of Computer Science, Bar Ilan University, Ramat Gan. Israel; email: benny@pinkas.net.

DOI: http://dx.doi.org/10.1145/2998550

extracted features into a risk model that outputs a risk score, where a higher risk score indicates a higher risk that the log in was attempted by an imposter. When the risk score is greater than some threshold, the web server may deny access to the account or take additional measures to identify the user's identity.

Impersonation detection systems store historical login attempt data and generate risk scores based on this data. When the web server receives a login attempt that involves a user's account, the server inputs the features of this login attempt, and the user's historical login attempt data, into a risk model that outputs a risk score.

The storage and analysis of the historical data, as well as the computation and update of the risk model, are increasingly outsourced to security companies specializing in this task. A company of this type runs a third-party service which stores all data and sends the risk score to the bank's web server so that the latter may grant or deny access to the users' accounts.

Unfortunately, there are privacy risks and potential liabilities that come with storing sensitive information (such as historical login attempt data) in the raw form that is needed as an input into risk models. Along these lines, many clients of a third party impersonation detection service (e.g., the online bank) would rather not send sensitive information in raw form. Further, certain regulations prohibit the export of such sensitive information to third parties.

One way to address this issue is to have the client of the third party service encrypt the sensitive information before sending it to the third party service. However, while this approach might satisfy regulations, the risk models used in the conventional approaches would not be able to work with this encrypted information.

Our Contribution. We focus on impersonation detection based on a ground speed model (see details below). We describe an impersonation detection system that uses secure multi-party computation (MPC) for computing the risk score of login attempts. The system stores the historical login data in encrypted form at the third-party service, and runs the computation while hiding from the third-party service the historical and current login data, and hiding from the login server the details of the model used for the computation of the risk score. We implemented and tested the system with large scale real-life data of actual users.

1.1. The Setting

The setting includes parties of three types (depicted in Figure 1):

—The *server* is run by a service company specializing in risk assessment of login attempts. The server knows the exact risk model that is used. It can store historical login data in encrypted form but must not be able to learn the actual details of that data.

—The *client* has its own users that attempt to log in to its services. The client hires the services of the server in order to run a risk model for preventing illegitimate login attempts. It does not wish, however, to reveal the actual login data to the server.

The client will obviously need to store a key that is used for encrypting the data that it stores at the server. Moreover, the setting might include several access points that are operated by the same client. For example, the client may be a large bank with multiple front-ends to which users might log in. It is desirable that these multiple client access points do not have to continuously synchronize information between themselves. Indeed, in our system, these access points only need to store copies of the same key that is used for encrypting data at the server and do not need to run any synchronization protocol between themselves.

—*Users* attempt to log in to services of the client.

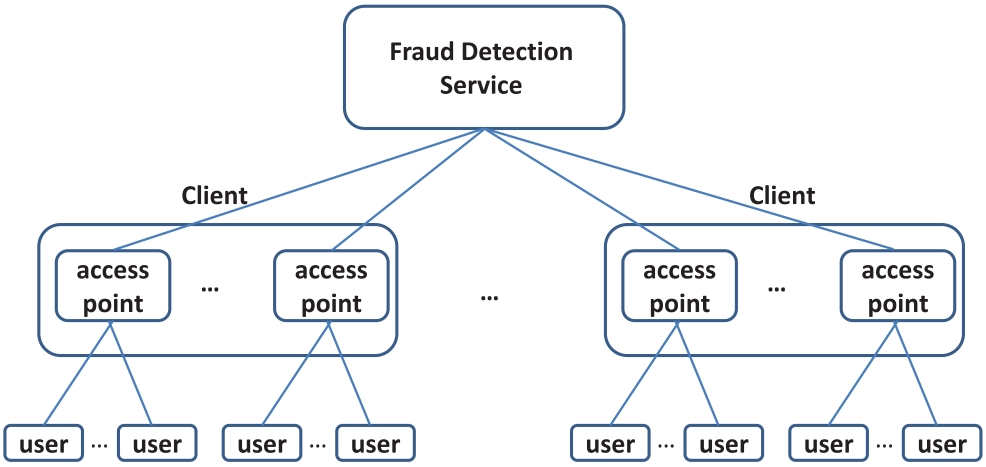

Fig. 1. The setting.

2. THE GROUND SPEED MODEL

The ground speed model is a simple risk model that is based on the following principle: There is a physical limit on the speed in which a user can move between locations (this speed is bounded by, say, the speed of a commercial airplane). Therefore, if two subsequent login attempts of the same user originate from locations that are too far from each other (namely, where moving from one location to the other requires an extraordinarily high speed), then the risk model reports a high-risk score.

There are, however, some issues that complicate the usage of the simple ground speed model:

—*Obtaining the location data:* The location of users must be input to the model. This information is extracted from different data that is available about the users. In particular, in this work, we assume that the geolocation data is extracted from IP addresses, although additional data can be extracted from other sources, such as GPS data or cellular network data.

—*Preventing false positives:* False alerts might be caused by different reasons. One very common event is a user who attempts to log in using different devices that are connected through different networks. For example, the user might use both her phone and her computer to simultaneously log in to a system. Each of these devices might be connected to the internet through a different network provider (e.g., a cellular network and an internet service provider), and the connection points of these providers to the rest of the internet might be located far away from each other. As a result, it might seem that the user is attempting to log in from locations that violate the speed restrictions of the ground speed model, and the risk model will, therefore, output a false alarm. A similar problem might occur with a user who uses a proxy or a VPN service.

In order to reduce the number of false alarms, the model is extended with the following change: Each IP address is associated with an "autonomous system" (AS), which is roughly the network provider operating this IP address. An AS is represented by an *ASname* and an *ASnumber*. The computation of the ASname and ASnumber is done using a public list (of a few hundred thousand entries) based on the IP address of the user. In addition, the model associates a country and hostname to each IP address. (Roughly speaking, the hostname is the a label assigned to the device the user is logging in from, while the ASname/ASnumber describe the name/number of the service provider.)

In order to reduce the number of false alarms, the model does not issue an alert if a user performs two successive logins with the same ASname, ASnumber, or hostname. Otherwise, if two successive login attempts are from the same country, then the risk factor is reduced by a preset factor. This functionality can be tweaked based on experimentation to limit the risk of false alarms.

2.1. The Model

We describe here in more detail the basic structure of the computation of the ground speed model.

Input. The input to the model is given by the client. It contains data associated with two successive login attempts of the same user. The client would like to keep this data hidden from the server. The data includes the following items:

—The time of the login attempt (denoted as time_b, where $b \in \{1, 2\}$ refers to either the first or the second login attempt).
—The latitude, longitude, and country of the geolocation (denoted as lat_b, long_b, and country_b).
—The hostname.
—The autonomous system name and number (ASname_b, ASnumber_b).

Parameters. The parameters in this list are known to the server computing the model. These parameters (except for the first parameter, the radius of Earth) might be proprietary information which the server would like to keep private and hide from its clients.

—The radius of Earth (R=6,371km).
—Minimum and maximum velocities (Vmin=0kmh, Vmax=815kmh).
—A distance error (DistError, which is set to be a few hundred km, based on a prior analysis done by the server).
—$\varepsilon = 0.0001$sec, and ConfidenceMin, which is set to a number in the range $(0, 1)$.
—ScoreThreshold $= 950$
—SameCountryFactor, which is set to a number in the range $(0, 1)$.

The Algorithm. We describe here a simplified algorithm that is used to set a risk score S. (The actual algorithm might be more detailed, but it follows the same lines.)

(1) Compute the distance according to the spherical law of cosines:

$$\text{Distance} = R \cdot \arccos(\sin(\text{lat}_1)\sin(\text{lat}_2) + \cos(\text{lat}_1)\cos(\text{lat}_2)\cos(\text{long}_1 - \text{long}_2)) \quad (1)$$

(2) Compute the confidence as

$$\text{Confidence} = \max\left(1 - \frac{\text{DistError}}{\text{Distance}}, 0\right) \quad (2)$$

(Note that if the distance is smaller than the distance error, then the confidence is 0.)

(3) Compute the speed as

$$\text{V} = \frac{\text{Distance}}{|\text{time}_1 - \text{time}_2| + \varepsilon} \quad (3)$$

(ε is added to the denominator to avoid dividing by zero when two logins occur at the exact same time.)

(4) Compute the risk score S:
  (a) If Confidence is smaller than ConfidenceMin, then $S = 0$.
  (b) If the user used the same ASname or the same ASnumber in both logins, then $S = 0$.
  (c) If the hostname (i.e., device) is the same in both logins, then S=0.
  (d) Else (conditions a,b, and c are false) set S to be $= 1000 \cdot \frac{\text{V}-\text{Vmin}}{\text{Vmax}-\text{Vmin}}$. (Note that if V > Vmax, then S > 1000.)
  (e) If both logins were from the same country (i.e., $\text{country}_1 = \text{country}_2$), then set $\text{S} = \text{S} \cdot \text{SameCountryFactor}$.
(5) If S > ScoreThreshold, then issue an alert.

3. SECURE COMPUTATION OF THE GROUND SPEED MODEL

3.1. Secure Computation

Secure multi-party computation enables two or more parties to compute any function of their private inputs while hiding everything about their inputs except for the desired output of the function. This problem was first investigated by Yao [1986] for the two party setting, and by Goldreich et al. [1987], Ben-Or et al. [1988], and Chaum et al. [1988] for settings with more than two parties. As an example, consider the "millionaires problem," where two millionaires, Alice and Bob, wish to find out which one of them is richer. They wish to do that without disclosing any other information to each other. Namely, if Alice has $\$X$ and Bob has $\$Y$, then they wish to compute a single-bit output, which is 1 if $X > Y$ and is 0 otherwise. They must do that without disclosing any other information about their input. (Obviously, the output itself reveals to Alice whether Bob's input is greater than hers, and vice versa, but the parties wish to reveal no other information about their inputs.)

The computation could easily take place in an ideal world, in which there is a third party that is trusted by both Alice and Bob. In that case, the two parties could send their inputs to the third party, and have it compute the function and send its output to them. The goal of secure multi-party computation is to achieve the same level of privacy and security without using any third party. This notion of security was rigorously defined (see, e.g., Goldreich [2004]).

There exist generic protocols for efficient secure computation of any function in both the two-party and multi-party settings. Our main focus in this work is in the two-party setting, for which the first secure protocol, suggested by Yao [1986], is still the basis for the most recent and most practical protocols for this task. Yao's protocol works by representing the function as a Boolean circuit, and then applying a simple secure computation technique for each gate of this circuit. It is composed of the following steps:

—The two parties agree on a representation of the function as a Boolean circuit.
—One of the parties, denoted as the *constructor*, constructs a version of the circuit which hides all information about the values of the internal wires of the circuit. It computes a small table for each gate of the circuit and sends these tables to the other party, denoted as the *evaluator*.
—For each bit of the evaluator's input, the two parties run a short protocol, known as *oblivious transfer*.
—At this point, the evaluator has sufficient information to compute the output of the circuit. The computation process hides from both parties all of the values of the internal wires of the circuit.

A detailed description of Yao's basic protocol appears in Lindell and Pinkas [2009]. There exist many improvements to Yao's protocol, and currently, it can support efficient computation of circuits of even billions of gates.

We will not discuss in this text the details and exact performance of the most efficient versions of Yao's protocol. As a coarse measurement, we note that currently secure computation of the AES encryption algorithm, which is represented by a circuit of about 30,000 gates, takes about 15msec [Gueron et al. 2015]. This corresponds to an impressive throughput of processing about 2M gates per second, but there are different details and caveats that affect achieving the same throughput for other computational tasks:

—The AES computation results that are quoted are for two parties that are located in the same data center.
—The results were achieved using the "free-XOR" method [Kolesnikov and Schneider 2008], which performs a secure computation of XOR gates with almost no overhead. The circuit computing the AES function has about 23,000 XOR gates and 7,000 non-XOR gates, and therefore, benefits a lot from the free-XOR method. Other functions might have boolean circuit representations, which have a much smaller fraction of XOR gates.
—These performance results were achieved for a rather small circuit. The computation of much larger circuits encounters issues such as not being able to store all data or all gates in the cache or in main memory. Such a computation is therefore harder to implement, and it might achieve a much smaller throughput.
—The AES circuit has a relatively small input. Secure computation applies an oblivious transfer protocol to each input bit (whereas other gates are computed using symmetric cryptographic operations that are more efficient). Functions with larger inputs will have worse performance.
—On the other hand, the result we quoted was achieved using a single core and communication via a commercial network. Using more cores and a dedicated network can improve performance.

Given these issues, we can conclude that secure computations of small- or medium-sized circuits can be extremely efficient, but the secure computation of very large circuits might be slower, and is definitely harder to implement. In general, secure computation is more efficient for circuits that are smaller, have a larger fraction of XOR gates, and have fewer input bits.

Naive Computation of the Ground Speed Model using Secure Computation. A direct approach for computing the ground speed model using Yao's protocol will have one party, the client, input to the protocol the current login information, and the other party, the server, input the historical data as well as its set of parameters. There are two problems with this approach: First, the size of the server's input. Furthermore, the computation includes computing the distance function and the velocity function, and these functions use computations of trigonometric functions and of divisions, which are hard to express using a Boolean circuit.[1] As a result, the computation might not be sufficiently efficient, in particular, when having to handle peak access loads. We, therefore, use a more refined approach for applying secure computation to the computation of the ground speed model.

3.2. Requirements

There are important operational requirements that affect the design of the system.

[1]The asymptotic size of such a circuit would be polynomial, but the actual size of the circuit would be pretty big. For example, in order to compute the five trigonometric functions that are used for computing the distance, the circuit would need to use a Taylor approximation, such as $\sin(x) = x - x^3/3! + x^5/5! - \cdots$. Each multiplication needs a sub-circuit of size which is quadratic in the length of the inputs, whereas the circuit that we use is linear in the size of the inputs.

The Model and the Parameters. The model and the parameters are the proprietary information of the server running the risk assessment service. It would like to leak as minimal information as possible about them to its competitors. Furthermore, the server might need to update the model and the parameters on an ongoing basis, based on observations and analysis of actual threat patterns. Therefore, the computation must be jointly computed by the server rather than by sending the risk assessment software to the client and running the software at the client site.

Storing Historical Data. Naively, historical login data about a specific client of the service could be stored at the client. However, clients typically prefer to store this data at the server for operational reasons: Firstly, the client is not in the business of storing this data. Since it happens that the historical login data is important for risk analysis, the client might prefer outsourcing the storage of this data to the external server running the risk analysis rather than buying and maintaining the hardware and software required for storing the data. Moreover, the client (e.g., a bank) might operate many access points to which users can log in. Each of these access points might need to contact the server in order to run the risk model. The historical data must be a unified view of the logins to all of these access points. The task of continuously synchronizing this information between the access points at the client side might be too difficult.

In order to solve this issue, the historical data is kept encrypted at the server, where the encryption is performed using a key that is known to each of the access points of the client. This enables each access point to read and update the historical data.

3.3. The Basic Principles of the Secure Computation Solution

The secure computation of the ground speed model is based on the following basic principles:

—*Historical data stored at the server:* The client stores encrypted historical data at the server. The encryption is a symmetric encryption using a key which is known to the client (and to all its access points), but not to the server.

For some data fields, such as country name, host name, ASname, and ASnumber, the client does not store the encrypted fields but rather a Message Authentication Code (MAC) of these values, computed using a key known only to the client. Moreover, the client only stores an ℓ-bit suffix of these MAC values, where ℓ is a parameter.

The reason for storing a short MAC is that these values are only used in the model in equality checks with the values of these parameters in the previous login attempt. Therefore, it is sufficient to store and use ℓ bits suffixes of these values, where ℓ is chosen so that the probability of equality between the ℓ-bit suffixes of the MACs of different values is sufficiently small.

The main advantage of using short ℓ-bit strings instead of, say, 160-bit long MAC values, is that these values are input to the secure computation phase and it is, therefore, preferable to minimize their size in order to make the secure computation more efficient.

—*Using different salts:* The encryption must hide from the server whether any data field (such as country, AS, latitude, etc.) changed between successive login attempts. In order to support this property, the encryption and MAC computations must be randomized. The encryption is in Cipher Block Chaining (CBC) mode with a randomly chosen IV. For the MAC computations, the client chooses a new salt whenever it computes new MAC values that need to be stored at the server. This salt value is also stored at the server and can be retrieved by the client together with the encrypted data.

—*Offloading part of the computation to the client:* The computation of the distance and speed in Equations (1) and (3) is arguably the most mathematically complex part of the secure computation since it uses trigonometric functions and division. However, this part of the computation is only based on information that belongs to the client, namely the location and time of login attempts, and uses known equations for the computation. There is, thus, no need to hide this computation from the client. Therefore, the client retrieves the encrypted historical information that is stored at the server and uses the previous and current location and time data in order to compute the distance and speed. It then inputs the result of this computation to the secure computation of the risk score.

—*The actual secure computation* The actual secure computation is done using Yao's protocol, which is applied to a circuit computing the risk model. The protocol is run between the server and the client, where the server is the constructor of the circuit and the client is the evaluator of the circuit. The client, therefore, learns the output of the computation.

3.4. Security

The common practice of working with secure computation requires defining the function that needs to be computed and proving that the process of running the secure computation is equivalent to using a completely trusted party who receives the inputs of the two parties, performs the computation, and sends out the outputs. Indeed, if we were using a generic secure computation protocol, such as Yao's protocol, for computing the entire risk model function, this would have been the straightforward way of defining and proving security (proving security in this case is actually trivial, since Yao's protocol is known to be secure according to this model [Lindell and Pinkas 2009]).

The structure of our protocol is more complex due to the requirement of storing historical client data at the server (this requirement was communicated to us by stakeholders in this domain and was defined as a "must"). To satisfy this requirement, the client stores encryptions of the historical data at the server. When the client wishes to compute the risk score, it retrieves this data and then stores it at the server re-encrypted (using fresh randomness). The actual computation of the risk score is applied using Yao's protocol.

We do not provide here a detailed cryptographic proof of security (for lack of space and lack of relevance to the audience of this special issue). The proof must show that none of the parties learns in the protocol more than it learns in an interaction with a trusted party. The protocol contains three interactions: the client retrieving the encrypted data; the two parties running a Yao computation of the risk score; and the client storing re-encryptions of the (possibly modified) data. It is straightforward to prove that if the encryption function is secure, and since each encryption uses fresh randomness, then the server cannot learn the stored encrypted data or identify whether this data has changed. The actual computation of the risk score is done using Yao's protocol and, therefore, it follows from Lindell and Pinkas [2009] that this computation leaks nothing more than a computation of the same function by a trusted party.[2]

Why run a secure computation? The input of the secure computation contains user data that is known to the client and some parameters known to the server. One might claim that the parameters are not very secret and, therefore, the server could just send them in plaintext to the user and have the user run a plain computation of the

[2]A subtle issue, which we do not address, is that the protocol gives each user a fixed pseudonym, and, therefore, the server can identify the times in which the client computes the risk scores of each (anonymized) user. This seems inevitable unless we hide from the server which historical data is retrieved for each computation. (It is possible to hide the identity of the retrieved data using private information retrieval (PIR), but this will incur a much higher computational overhead for the server. In particular, for each data retrieval, the server will have to apply cryptographic operations to each data item that it stores.)

risk model. However, one of the specific requirements that instructed the design of the system was that the parties must run a *joint* computation. This is needed in order to enable the server to provide easy updates to the computed function, control its operation, and be able to measure the usage of the system.

Privacy of the function: The security model, and Yao's protocol, do not prevent the client from learning the code of the function that is computed (namely, the algorithm used for computing the ground speed model). This was not a design goal of our system. If the server wishes to keep this function secret, it can use *private secure computation* in order to hide it. This can be done by applying Yao's protocol to a universal circuit which hides the wiring between the gates at the cost of increasing the overhead by a factor of $O(\log n)$, where n is the number of inputs (see, e.g., Mohassel and Sadeghian [2013]).

3.5. The Secure Computation Protocol

Notation. We use the notation $\text{AES}_K(\text{m})$ to denote a single application of AES. We write AES-CBC_k to denote a CBC encryption using AES with a random IV (this is an operation mode of AES that is used for encrypting messages longer than 128 bits). We write $\text{HMAC}_k(\text{m})$ to denote HMAC authentication of a message m using a key k and the hash function SHA2.

We use a parameter ℓ which defines the probability of obtaining a false positive in the checks for equality in the protocol. We set $\ell = 32$, which sets the false alarm probability to $4 \cdot 2^{-32}$.

Input to Client. The client has a master key K, which is used in order to generate two other keys:

—K1 = AES_K("AES"). (This key will be used for AES encryption).
—K2 = AES_K("HMAC"). (This key will be used for HMAC computations).

When a specific user attempts to log in, the client obtains the following input values associated with that user:

—user pseudonym ID (this is a random pseudonym for the user that is kept the same throughout the usage of the system).
—time1 (the time of the recent login)
—lat1, long1 (the location coordinates of the recent login)
—country1 (the country from which the recent login took place)
—hostname1 (the host name of the recent login)
—ASname1, ASnumber1 (the AS information of the recent login)

Input to Server. The server stores the following values associated with each specific user of the client:

—user pseudonym ID
—salt
—c1 = AES-CBC_{K_1}(time2 | lat2 | long2)
—c2 = The ℓ least significant bits of HMAC_{K2}(salt | country2)
—c3 = The ℓ least significant bits of HMAC_{K2}(salt | hostname2)
—c4 = The ℓ least significant bits of HMAC_{K2}(salt | ASname2)
—c5 = The ℓ least significant bits of HMAC_{K2}(salt | ASnumber2)

The Protocol. The protocol contains the following steps:

(1) The client sends the value of ID to the server.
(2) The server sends to the client the salt and c1 value of the corresponding user.
(3) The client decrypts c1 using the key K_1 and obtains time2, lat2, and long2.
(4) The client computes the distance D, the confidence Conf, and the speed V, as in the ground speed model, in the following way:
  (a) $\text{Distance} = R \cdot \arccos(\sin(\text{lat}_1)\sin(\text{lat}_2) + \cos(\text{lat}_1)\cos(\text{lat}_2)\cos(\text{long}_1 - \text{long}_2))$.
  (b) $\text{Conf} = \max\left(1 - \frac{\text{DistError}}{\text{Distance}}, 0\right)$.
  (c) $\text{V} = D/(|\text{time}_1 - \text{time}_2| + \varepsilon)$.
  (d) S = V·1.227 (this constant results from setting Vmax=815, Vmin=0 in the ground speed model).
  (e) S = 4· S (this multiplication is done in order to verify that S is divisible by 4).
(5) The client computes the following values
  —c'2 = The ℓ least significant bits of HMAC_{K2}(salt | country1)
  —c'3 = The ℓ least significant bits of HMAC_{K2}(salt | hostname1)
  —c'4 = The ℓ least significant bits of HMAC_{K2}(salt | ASname1)
  —c'5 = The ℓ least significant bits of HMAC_{K2}(salt | ASnumber1)
(6) The two parties run Yao's secure computation protocol computing the following circuit:

  (a) The inputs to the circuit are values c'2,c'3,c'4,c'5,Conf,S (known to the client) and c2,c3,c4,c5 (known to the server).
    Note that all inputs are integers, except for Conf, which is in the range [0,1). This value is encoded in the input as a 32-bit unsigned integer, which represents the value Conf·2^{32} (namely, it encodes the value of Conf in 2^{-32} increments).
  (b) The server plays the role of the constructor in Yao's protocol, and the client plays the role of the evaluator. The output of the circuit is learned by the client.
  (c) The circuit computes the following function:
    (i) If (Conf < 0.75) OR (c4 = c'4) OR (c5 = c'5) OR (c3 = c'3) then S = 0.
    (ii) S = MIN(S,4000).
    (iii) If (c2 = c'2) then S = 0.75 · S (this operation is implemented by shifting S to the right one and two positions, and adding the two resulting values).
    (iv) Output S.

  In order to ensure that the multiplication by 0.75 results in an integer value, S is always multiplied by four before being input to the circuit.

  This circuit could be replaced by the server with a different circuit whenever the ground speed model or the parameters are changed.
(7) After receiving the output S of the circuit, the client performs the following operations:
  (a) S = S/4.
  (b) If S > 950, then issue an alert.
(8) (Prepare information for next login) The client chooses a random string, denoted newsalt, and computes the following values:
  —c"1 = AES-CBC_K(time1 | lat1 | long1)

—c”2 = The ℓ least significant bits of HMAC_{K2}(newsalt | country1) (This step and the next steps are the exact computation as in the items of Step 5, but using the value newsalt rather than salt.)
—c”3 = The ℓ least significant bits of HMAC_{K2}(newsalt | hostname1)
—c”4 = The ℓ least significant bits of HMAC_{K2}(newsalt | ASname1)
—c”5 = The ℓ least significant bits of HMAC_{K2}(newsalt | ASnumber1)
These values will now be stored at the server and will be used by the client in the subsequent login attempt of the same user. The MAC values are computed using a new salt value (newsalt) in order to ensure that the server cannot identify whether any of the values (country, hostname, ASname, ASnumber) have changed between login attempts. This prevents the server from identifying whether the user logged in from identical locations or had other repeating values.

(9) The client sends the values (newsalt, c”1, c”2, c”3, c”4, c”5) to the server.
(10) The server stores (ID, newsalt, c”1, c”2, c”3, c”4, c”5) for usage in the next login attempt of the same user.

4. EXPERIMENTAL RESULTS

The protocol is composed of two basic parts: the computations, encryptions, and decryptions that are performed independently by each party; and the secure computation of the circuit using Yao’s protocol. We implemented both parts of the protocol and measured its runtime. We next describe in more detail the construction of the circuit that is used in Yao’s protocol. This circuit was optimized in order to improve the performance of the secure protocol.

4.1. The Circuit

The circuit computing the required functionality was constructed by a Java program which computed the gates and inter-connections of the circuit. This program is run once for any choice of parameters. The output of the program is an HDL-like description of a circuit which is used by the programs implementing Yao’s protocol. We used a program for writing the circuit, rather than designing it by hand, in order to easily support changes to the parameters used for defining the computation.

The construction of the circuit depends on the following parameters:

—The length of the Conf value that is used in the protocol (in our experiments Conf was a 32-bit binary number).
—The size of value S (in our experiments it is set to be 16 bits long).
—The size of the values c2,c3,c4,c5 (in our experiments, each of these values is set to be 32 bits long).
—The constant value that is used in the MIN calculation in Step 6.4.ii of the protocol (in our experiments, it is to be 4,000).

The construction of the circuit attempted to optimize its size. We will describe here two optimizations that might be useful for other protocol designers:

—Step 6.4.ii of the protocol computes the minimum between S and a constant (which is equal to 4,000 in our experiments), and assigns the result to S. This computation performs a comparison of two numbers and then computes a multiplexer which sets to S one of two values based on the result of the comparison. There are known circuits for computing comparisons and multiplexing, but in our protocol, it is possible to utilize the fact that one of the two numbers that are compared is a constant, and

construct a much smaller circuit for the task. The simplest way of constructing the circuit is by starting with known circuits comparing and multiplexing two *variables*, and then fixing the input bits representing the constant input bits and removing gates whose values are set when these bits are set. (For example, an AND between an unknown bit x and a constant bit can be replaced with a 0 if the constant bit is 0, or with x if the constant bit is 1.) Further optimizations are also possible.
—Step 6.4.iii of the protocol multiplies S by 0.75. In general, implementing multiplications by Boolean circuits requires rather large circuits. However, in this case, the multiplication can be easily implemented by shifting S one bit to the left (corresponding to a multiplication by 0.5), shifting S two bits to the left (corresponding to a multiplication by 0.25), and adding the two results.

For the choices of parameters that we made, the resulting circuit had 454 gates, of which 188 gates are XOR gates. (Note that secure computation with the free-xor optimization [Kolesnikov and Schneider 2008] can compute XOR gates with essentially no overhead. It is, therefore, preferable to maximize the fraction of XOR gates in the circuit.) The first party inputs 128 bits to the circuit, whereas the second party has a 176-bit input. The size of the circuit is, therefore, very close to the number of input bits (which is an obvious lower bound for the size of any circuit computing a function, which depends on all input bits).

4.2. Performance

We implemented all parts of the protocol using the SCAPI library for secure computation [Ejgenberg et al. 2012]. In order to make the computation as efficient as possible, we implemented the secure computation using the following optimizations:

—The oblivious transfers were computed using OT extension [Asharov et al. 2013].
—The protocol used the free-xor optimization of Kolesnikov and Schneider [2008].
—The gates were encoded as tables of size 3, using the garbled row reduction optimization [Pinkas et al. 2009]
—The encryptions in the gates were computed using the AES-NI Intel hardware instruction and a fixed AES key.

We implemented the roles of both parties, and ran both parties on the same machine, running Windows on an Intel i5 3.2GHz CPU with an 8GB RAM. We repeated the run of the protocol 100 times, and computed the runtime of each execution. The average runtime of an execution of the protocol was 3.085msec, with a variance of 1.17msec. Of this time, about 2msec was taken by the execution of Yao's secure computation protocol, and the rest of the time was used for encryptions and decryptions, and preparations for running the secure computation protocol.

Our experiments were run on a single machine, but obviously the client and server will be using different machines. The protocol requires five messages (two for the initial retrieval of the stored historical data, two for running Yao's protocol, and one for storing updated historical data). Latency inside a data center is typically smaller than 1msec, while the latency within North America, Europe, or between Europe and the east coast is less than 45ms, 30ms, and 90ms, respectively. (See http://www.verizonenterprise.com/about/network/latency/, retrieved April 2016.) This means that, except for the case where the client and server are located in the same data center, the communication latency dominates the latency induced by the computation. (Note also that the client can make a decision about the risk factor after the completion of Yao's protocol, in parallel to the last step of sending the re-encrypted data to the server.)

The system is designed in a way which makes the runtime of the cryptographic protocol independent of the number of clients and the number of users. Namely, the number of clients and the number of users only affect the time that it takes the server to retrieve the relevant encrypted records from the its storage, which should be negligible. Afterward, the cryptographic decryption and Yao's protocol are always applied to the data of a single user.

5. CONCLUSIONS

This work describes the computation of an existing risk score model while preserving the privacy of the data and of the parameters used in the model. The scale of the data and the complexity of computing the mathematical functions used in the model prevent a straightforward implementation of a generic secure computation protocol. Instead, our work examined the privacy requirements and delegated computation and storage tasks to both participants in a way that preserves privacy and enables an efficient computation.

ACKNOWLEDGMENTS

We would like to thank Meital Levy for implementing the protocol, measuring its performance, and discussing it with us; and thank Yehuda Lindell for fruitful discussions about this work.

REFERENCES

Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. 535–548. DOI:http://dx.doi.org/10.1145/2508859.2516738

Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM, 1–10.

David Chaum, Claude Crépeau, and Ivan Damgard. 1988. Multiparty unconditionally secure protocols. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. ACM, 11–19.

Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. 2012. SCAPI: The Secure Computation Application Programming Interface. Cryptology ePrint Archive, Report 2012/629. (2012). http://eprint.iacr.org/.

Oded Goldreich. 2004. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY.

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. ACM, 218–229.

Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. 2015. Fast garbling of circuits under standard assumptions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 567–578. DOI:http://dx.doi.org/10.1145/2810103.2813619

Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming*. Lecture Notes in Computer Science, Vol. 5126. Springer, Berlin, 486–498.

Yehuda Lindell and Benny Pinkas. 2009. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology* 22, 2 (2009), 161–188.

Payman Mohassel and Seyed Saeed Sadeghian. 2013. How to hide circuits in MPC an efficient framework for private function evaluation. In *Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'13),* Thomas Johansson and Phong Q. Nguyen (Eds.), Vol. 7881. Springer, 557–574. DOI:http://dx.doi.org/10.1007/978-3-642-38348-9_33

Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. 2009. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'09),* Mitsuru Matsui (Ed.), Vol. 5912. Springer, 250–267. DOI:http://dx.doi.org/10.1007/978-3-642-10366-7_15

Andrew Yao. 1986. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*. IEEE, 162–167.

Received November 2015; revised April 2016; accepted September 2016