Circuit-based PSI for Covid-19 Risk Scoring

Leonie Reichert*, Marcel Pazelt*, Björn Scheuermann*†
*Humboldt University of Berlin, Department of Computer Science
{leonie.reichert, marcel.pazelt, scheuermann}@informatik.hu-berlin.de
†Alexander von Humboldt Institute for Internet and Society, Berlin

***Abstract*—Many solutions have been proposed to improve manual contact tracing for infectious diseases through automation. Privacy is crucial for the deployment of such a system as it greatly influences adoption. Approaches for digital contact tracing like Google Apple Exposure Notification (GAEN) protect the privacy of users by decentralizing risk scoring. But GAEN leaks information about diagnosed users as ephemeral pseudonyms are broadcast to everyone. To combat deanonymisation based on the time of encounter while providing extensive risk scoring functionality we propose to use a private set intersection (PSI) protocol based on garbled circuits. Using oblivious programmable pseudo random functions PSI (OPPRF-PSI) , we implement our solution CERTAIN which leaks no information to querying users other than one risk score for each of the last 14 days representing their risk of infection. We implement payload inclusion for OPPRF-PSI and evaluate the efficiency and performance of different risk scoring mechanisms on an Android device.**

***Index Terms*—Digital Contact Tracing, Private Set Intersection, OPPRF-PSI, Risk Scoring**

I. Introduction

Covid-19 has influenced our daily lives since it reached the scope of a global pandemic in the beginning of 2020. To lower the number of infections in the population several controlling measures such as quarantine, social distancing, mask mandates, regional lockdowns and contact tracing have been employed. The latter aims at identifying possible infection chains through tracing individuals who have been in contact with a diagnosed patient. Manual contact tracing efforts are difficult to maintain with the large amounts of new cases. Therefore *digital contact tracing* (DCT) systems are being employed to support manual efforts by crowd-sourcing the task. For this purpose, mobile apps are used to identify close contacts with other app users and notify their bearers about their risk of exposure. To be effective, such an app has to register proximity events in a reliable way, provide the user with a realistic exposure risk estimation and has to be widely used. In most regions participation is voluntary. Therefore, secure solutions with good privacy protection are needed to ensure acceptance and adoption. Modern cryptographic protocols from the field of *secure multi-party computation* (MPC) can help to develop a privacy-preserving DCT app with minimal risk of leaking information about the participants.

In this paper, we develop a system for DCT called CERTAIN which uses the circuit-based protocol of Pinkas et al. [1] for *private set intersection* (PSI) based on *oblivious programmable pseudo-random functions* (OPPRF). Our contributions are:

- Designing and evaluating an approach to DCT using OPPRF-PSI with an Android app
- Implementing an extension for OPPRF-PSI which allows the inclusion of payload from client and server
- Designing complex risk scoring for DCT using circuit-PSI with payload
- Evaluation of OPPRF-PSI for unbalanced sets with parameters aligned to the context of DCT

We test a variety of circuits that align with existing DCT risk scoring functionalities. The system was evaluated in regards to communication, runtime and energy efficiency in the context of DCT for different networks.

In Section II, we introduce relevant related work on DCT. Next, Section III describes the OPPRF protocol and our system, as well as adjustments that had to be made to realize risk scoring for DCT. Nuances of the implementation are explained in Section IV. The evaluation is presented in Section V, followed by a discussion in Section VI. We conclude in Section VII.

II. Related Work

Contact tracing aims at identifying people who have been infected with a monitored disease by analyzing who has been in contact with a diagnosed person. If new cases are discovered and isolated fast enough, the spread of the disease can be limited. As Covid-19 is a quickly spreading, airborne disease with pandemic scale, crowd-sourced contact tracing has become increasingly relevant since the start of 2020 [2]. Evaluations based on observations, for example [3] conducted by Wymant et al. focusing on England and Wales, have found that these DCT apps had a positive impact on the course of the pandemic. DCT relies on modern technology to detect which people had been in each others presence for a significant amount of time so that a transmission is likely if either person turn out to be infected later. Smartphones are a convenient tool to this end as many people continuously carry their device on them and keep them turned on. The various technologies available in off-the-shelf smartphones can be used to detect relevant encounters. During the last year, Bluetooth Low Energy (BLE) has emerged as most practical technology for this purpose [2].

DCT approaches using BLE rely on users exchanging *ephemeral pseudonyms* (EPs) over BLE with others close by to register encounters. The most widespread example is the

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

Google Apple Exposure Notification (GAEN) framework [4]. It provides the foundation on which many national DCT applications, such as the German Corona-Warn-App [5], are built upon. In GAEN, EPs are derived from a daily key. Newly diagnosed users upload their recent daily key to the server of the local *health authority* (HA) from where they are distributed to all users. Users will check locally if they have recently recorded any of the EPs that can be derived from the daily keys published by the HA. They are warned through the app if in doing so an encounter with an infected person is detected. Similar to manual contact tracing efforts, the risk of infection for encounters with infected users is estimated. In GAEN, so-called exposure scores are calculated according to duration and closeness of the encounter with the infected person, as well as their infectiousness.

In a system like GAEN where EPs of diagnosed users are published and risk scoring is conducted locally, a malicious user who received a warning can **deanonymize the corresponding infected individual** by remembering who they have been in contact with during the time of the encounter. One approach to solve this issue could be to ensure that a warning is delivered but no information is leaked about the time of the encounter. While this is possible, such a system is prone to attackers that attempt to deliver false warning [2]. To ensure that users only receive legitimate warnings but do not learn more about the infected person who caused the warning, cryptographic approaches can be used.

MPC allows two parties to evaluate a joint function $y_1, y_2 = f(x_1, x_2)$ over the inputs of the participants. The inputs remain secret from the other participant and each party i only learns the final result y_i. Any function f that is solvable in polynomial time can be represented as an MPC protocol [6, Chapter 22.2]. One way to realize MPC protocols are *Yao's garbled circuits* [6]. Here, one side creates a *circuit* from the function to be calculated and send it to the other party, which evaluates the circuit. Evaluation requires oblivious communication between both parties. Approaches to DCT that use MPC mostly use PSI. In these systems, the HA's server stores past EPs of diagnosed individuals. Users individually contact the HA's server to derive their risk and use recorded EPs for their query.

Both sides input their data to the secured communication. The user will learn their risk of infection, but they will not learn information about the server data set. Berke et al. [7] use Diffie-Hellman PSI to determine the size of the intersection. The approach of Demirag et al. [8] provides the querying user with the size of the intersection. Trieu et al. [9] use a form of Diffie-Hellman PSI optimized for asymmetric set sizes. The Catalic system proposed for DCT by Duong et al. [10] reduces the PSI load on querying client devices by shifting computation to servers that do not have to be trusted. In one of our earlier works [11], we used binary search on an oblivious RAM for DCT. Unlike the approach proposed in this paper, these systems do not compute a risk score but only return the intersection to the user, which results in privacy issues similar to GAEN, or the cardinality of the intersection, which are difficult to interpret.

III. DCT with OPPRF-PSI

In the following we describe the attacker model and explain how CERTAIN uses circuit-based PSI to implement complex risk scoring for DCT.

A. Attacker Model

In the context of MPC there are two types of attackers who can participate in a protocol. A *semi-honest* adversary is interested in learning as much as possible during the computation about secret data of the other party while still following the protocol. A *malicious* adversary might deviate from the protocol and try to pass incorrect data to the computation.

To understand the security requirements for the system in general, we also discuss several threats to DCT. *Evil users* of an DCT system might try to generate false alarms to convince other users that they are at risk. They can also try to deanonymize diagnosed users, meaning users that are verified Covid-19 cases. Another objective of an evil user can be attacking the availability of the system and thereby blocking the distribution of warnings, i. e., a denial of service (DoS). An evil user can behave either like a semi-honest or a malicious adversary during the computation. The *health authority* (HA) is a government branch responsible for handling the pandemic. It conducts tests and manages contact tracing efforts. Thus, the HA is interested in identifying and quarantining infected patients. An overambitious HA can try to deanonymize people at risk. As a government entity it might also have an interest in creating a database not only containing epidemiologically relevant data but all information about users it can gather. Such a database could later be used by other governmental bodies such as law enforcement. We assume the HA has no interest in breaking contact tracing by providing false information. An *eavesdropper* will try to identify users participating in the system. They are capable of observing the network traffic and/or collecting BLE beacons. Eavesdroppers can be network providers or entities with a number of widely deployed Bluetooth scanners, such as billboard companies. Additionally, by observing metadata like message sizes or frequencies, this attacker can try to derive infection statuses of users.

B. System Design

To participate in DCT, users download an app to their smartphone which regularly emits BLE advertisements. These advertisements contain an EP which is changed every 15 mins. The app also collects EPs of other users in the vicinity. To determine a distance to a sender, the signal strength is recorded. If a user tests positive for Covid-19, they pass their used EPs of the last 14 days to the HA. To warn users who have come into close contact with diagnosed users and are likely to have been infected, individual risk levels are computed daily. For each of the last 14 days, the user initiates a MPC computation with the server of the HA to calculate a risk score using OPPRF-PSI. For each computation the user inputs the EPs they recorded for that day while the server inputs the relevant EPs of diagnosed users. After the computation, the user will receive a set of risk scores. If the calculated risk for a day does not exceed a certain threshold, no risk is indicated. Otherwise, the actual risk score is revealed. Using these scores, the users can decide themselves if they need to follow the HA's directions and get tested. Since we assume this system is voluntary, reporting users who are at risk of being infected seems counterproductive. Therefore, risk scores are not revealed to the HA.

C. Circuit-Based PSI

In the following, we will first explain the used PSI protocol in general and then describe the function of an OPPRF.

1) PSI Protocol: The protocol we use for PSI is called OPPRF-PSI and designed by Pinkas et al. [12]. Unlike non-circuit PSI protocols, which are often more efficient, it allows to compute arbitrary functions on the output of the intersection without revealing intermediate results and additional information. Compared to other circuit-based PSI protocols like [13]–[15], it is the first to provide linear circuit complexity and a runtime of $\mathcal{O}(n)$.

For OPPRF-PSI, the server uses simple hashing to place their element set Y in β bins. The client uses cuckoo hashing [16] with the same hash functions on their set X to create a distinct match from elements to bins. The number of bins β is selected so that no stash is required during cuckoo hashing. The protocol uses a batched OPPRF sub-protocol to determine if a client's element occurs in the corresponding server side bin. This requires only one comparison. To this end, the server samples a set of target values $t^* = t_1 \cdots t_\beta$ from a random distribution. Both sides then invoke a batch-OPPRF using the hashing bins of both sides and the servers target values. This protocol phase returns a vector y^* of size β to the client which has the following property. If an element located in the client's bin j is contained in the server set, the value at the vector's position j is equal to the server's target value t_j, otherwise it is zero. Since communication is masked, the client can not tell target values from zeros. In the next step, both sides compute a circuit with a MPC protocol inputting X, y^* and t^*. The circuit compares for each position j if y^*_j is equal to t^*_j with an AND gate. For this purpose only γ bits are required to ensure fast runtime and a low false positive rate. Using all elements from X for which this test was successful the desired function f (e.g. a risk scoring function) is computed obliviously. The result is then revealed to the client.

2) OPPRF: As mentioned above the OPPRF takes hashing bins from both sides and returns a vector y^* to the client. To explain OPPRF, we first need to introduce *pseudo-random functions* (PRF). We write them as $F_{PRF}(k, q)$. For a randomly chosen key k and an arbitrary input q, the PRF outputs values which are indistinguishable from random elements in the function range. A PRF can be implemented as MPC protocol and is then called *oblivious PRF*.

An extension of normal PRF are *programmable PRF* (PPRF). On a certain "programmed" set of inputs X the PPRF outputs "programmed" values T , where $|X| = |T|$ and each $t \in T$ is uniformly distributed. When constructing the PPRF a *hint* is generated in form of a polynomial that maps each programmed input value $x \in X$ to the XOR combination of $F_{PRF}(k, x)$ and its target value t. To answer a query q, the hint is evaluated. If the PPRF was programmed at position $x = q$ then the query returns the corresponding target value t.

An *oblivious PPRF* (OPPRF) provides the functionality of a PPRF as two-party MPC protocol. Instead of a PRF an oblivious PRF is required. To improve performance, an OPPRF can be batched by running a number of β OPPRF instances on different parts of the input. The hints for all β instances can be combined into one.

D. Payload Inclusion

Payload data is private data associated with an element used by either side for calculating the intersection. Payload is used to compute functionalities for the intersection. To be able to include payload data, some additional implementation is needed. For payload inputs from the client, the adjustment is straightforward. The circuit has to be extended with input wires where the client inputs each element's payload. The result of $(y^* \wedge t^*)$ is combined by an additional AND gate with the client's payload. More work is required, when there is payload data originating from the server. In [12], Pinkas et al. explain the problem and give instructions on how to extend the basic protocol to allow the server to input payload data. They did not implement or evaluate this part themselves.

In the basic protocol without any payload the server maps multiple elements to each bin and the OPPRF assigns the same target value to all elements within a bin. If the circuit now detects a match between two bins of client and server it is impossible to infer which of the elements in the servers's bin had matched. To make this possible, the protocol requires two invocations of the batch-OPPRF. The first invocation is the same as in the original protocol. The second invocation is for identifying which payload of the elements that the server had mapped to a bin is related to the match with the client's element. As before, the client has input set X and the server has input set Y. Let $U(x)$ and $V(y)$ denote the payloads associated with $x \in X$ and $y \in Y$ respectively and let X_j be bin j of the client and Y_j of the server respectively.

The server chooses the target values such that the elements in each set T_j are not equal. Specifically, the client inputs its bins (like in the first OPPRF). The server samples $\tilde{t}_1, \cdots, \tilde{t}_\beta$ uniformly and inputs its own bins $Y_1, \cdots, Y_\beta$ and $T_1, \cdots, T_\beta$ where $T_j(i) = \tilde{t}_j \oplus V(Y_j(i))$ and $\oplus$ is the bitwise exclusive-OR operator. The batch-OPPRF outputs the result vector $\tilde{y}^*_j$ to the client. Now, the circuit computes for index j the following:

- The client inputs X_j, y^*_j, $\tilde{y}^*_j$ and $U(X_j)$. The server inputs t_j and $\tilde{t}_j$.
- The circuit compares y^*_j to t_j.
- If they are equal the servers payload has to be reconstructed. If X_j is the i-th item in server's bin Y_j, then the value received by the client is $\tilde{y}^*_j = \tilde{t}_j \oplus V(Y_j(i))$. Thus, $V(Y_j(i)) = \tilde{y}^*_j \oplus \tilde{t}_j$.

- Next, a sub-circuit computes the desired function f on X_j, $U(X_j)$ and $V(Y_j(i))$.

This payload inclusion adaption of the basic protocol results in the same asymptotic complexity. The circuit, which now handles payloads, computes the same number of comparisons as the basic circuit [12]. The actual duration of the OPPRF phase doubles since it is invoked twice on the same amount of bins.

E. Risk Scoring Circuits

Risk scoring is the calculation of an exposure score which reflects the risk of infection based on proximity events with diagnosed individuals. The first version of the GAEN API risk scoring approach [17] multiplies risk values for infectiousness r_i, duration r_d, days since exposure r_D and attenuation r_a per EP e from the set of pseudonyms of infected users D.

$$RS_{v1}^{GAEN} = \sum_{e \in D} r_{e,i} \cdot r_{e,d} \cdot r_{e,D} \cdot r_{e,a}$$

In the second version, changes in distance between users were also considered [17]. Here, duration at an attenuation range $j \in AR$ is multiplied with a corresponding weight w_j. The sum over all ranges is multiplied with a weight representing the infectiousness of the contact r_i and a value representing the reliability of the testing method r_{test}. GAEN defines four different attenuation ranges for immediate, near, medium and other encounters. GAEN leaves the task of defining exact dB values to the developers building upon its API.

$$RS_{v2}^{GAEN} = \sum_{e \in D} (\sum_{j \in AR} w_j \cdot r_{e,j,d}) \cdot r_{e,i} \cdot r_{e,test}$$

Similarly, the mechanism of DP-3T [18] multiplies exposure at three different attenuation ranges with static weights and then calculates a sum to determine the user's risk. The attenuation ranges are given by the thresholds 50dB and 55dB.

$$RS^{DP3T} = \sum_{e \in D} (\sum_{j \in AR} w_j \cdot r_{e,j,d})$$

These examples show that both summation and multiplication are relevant for complex risk scoring.

We evaluate several functionalities for risk scoring using OPPRF-PSI. The most simple mechanism calculates the sum of payload values provided by the client which belong to an element that appeared in the intersection. We call this functionality *AS*. The payload can be for example the number of minutes $r_{e,d}$ the user was exposed to a certain EP e. The circuit would then calculate the numbers of minutes for a day that the user was in contact with diagnosed individuals.

The next step is to allow both sides to provide payload used for summation. This circuit is called *AB*.

Our complex risk scoring allows to multiply payload values belonging to the same intersection element and then calculates the sum over all partial results. We call this circuit *ABM*. It allows the HA to provide information about the infectiousness for a specific EP. If the client uses $payload_A(e) = r_{e,d} \cdot r_{e,a}$ and the server inputs $payload_B(e) = r_{e,i} \cdot r_{e,D}$ as payload for each element e of their sets, a scoring model similar to the GAEN API v1 can be achieved. To produce full GAEN or DP-3T risk scoring which takes into account different attenuation levels, extra work by the client is required. For each recorded EP e it has to compute the following sum:

$$payload_A(e) = \sum_{j \in AR} w_j \cdot r_{e,j,d} \qquad (1)$$

The overhead for this computation is minimal, as it can be calculated 15 min after the EP was first received. For DP-3T risk scoring the server does not need to include any data. For GAEN v2 the server has to include $payload_B(e) = r_{e,i} \cdot r_{e,test}$.

We additionally evaluate a set of circuits, where a risk score is only revealed to the user if it exceeds a certain threshold. We apply this functionality to the three circuits described above, giving us the circuits AST, ABST and ABMT.

IV. Implementation and Parameters

To realize CERTAIN, we used a re-implementation of the OPPRF-PSI protocol [19]. It relies on the ABY framework [20] for circuit implementation. Using the Android Native Development Kit [21] we ported the OPPRF-PSI to Android by cross-compiling all dependencies. We improved the protocol to allow payload inclusion and implemented an Android app to conduct experiments.

A. Parameter Selection

For the implementation and evaluation, it is important to estimate the size of server and client set. DP-3T [18] assumes the number of diagnosed users who upload their data per day to be 2,000. The authors of Epione [9] use 5,000 daily cases for their evaluations. During the height of the pandemic in December 2020, 34,000 new daily cases were registered in Germany by the responsible authority [22]. It can also be assumed, that only a fraction of diagnosed people will have the app installed and will in case of an infection provide their EPs to the HA. In Germany, the Corona-Warn-App which builds on GAEN has been downloaded 28,3 million times as of June 2021 [5]. This is a dissemination of about 34,1%. Around 5,000 new cases per day who upload their data are considered for our system, which leaves room for greater outbreaks or a larger user base.

Various approaches to DCT use different durations for how long the same EP is advertised [2]. We assume that the infectious period is 14 days and EPs change every 15 mins. To be able to provide the user with per-day risk scores, the server set is split into 14 separate sets so risk scores can be computed for each of the corresponding days separately. Therefore, the server set size is set to $n_2 = 2^{19}$ which corresponds to 5461 daily cases. There are also some practical limits to the server set size. The implementation uses Lagrange interpolation in a prime field based on the Mersenne prime $2^{61} - 1$. This allows for operations like multiplication of field elements to be an order of magnitude faster, but it limits the field elements to be ≤ 61 bit long. Subsequently, the maximum amount of input elements is limited to $n = 2^{20}$ when requiring statistical security of $\sigma = 40$ and keeping the same protocol parameters as in the balanced set case of Pinkas et al. [12]. By adjusting the number of bins and batching parameters of the OPPRF phase we can allow the server set to be bigger. For the client we used $n_1 = 2^{10}$, as using a smaller set is irrelevant since the number of bins for client and server are influenced by maximal bin size on the server side.

B. Complexity

The OPPRF-PSI protocol has a linear asymptotic communication overhead in the number of elements. It has a complexity of $O(ln)$ gates, where n is the number of input elements and l their bit-length. The effect unbalanced sets have on the protocol complexity has not been discussed in [12], but each protocol phase is affected. Hashing complexities for client and server are asymmetric in this case. Due to the high difference in set size between server and client and restrictions on the maximal bin size, the number of bins β has to be scaled by a factor ρ. This results in a unbalanced set complexity of $O(l \cdot \rho \cdot \beta \cdot n_1)$ for the basic PSI circuit. For circuits, where both sides provide payload, the complexity is at $O(l \cdot \rho \cdot \beta \cdot n_1 + n_2 \cdot \delta)$ gates as payload inputs from the server with bit-length δ are included. For the OPPRF phase the server has to interpolate $O(K \cdot n_2)$ given K the number of hash functions used for cuckoo and simple hashing. The amount of data that is required to transfer the hint generated by the OPPRF for unbalanced sets is in $O(K \cdot n_2 \cdot \gamma)$. In the balanced setting Pinkas et al. noted that hints were just responsible for 3 % of the overall communication. As the data required for hint communication is in $O(n_2)$ and the basic circuit is in $O(n_1)$, this had to be re-evaluated for the unbalanced case. Also the impact of the OPPRF phase is doubled for circuits where the server also provides payload. The changes for asymptotic complexities suggest a substantial impact that a great increase of n_2 might have. To be able to discuss concrete performance of the protocol in an unbalanced setting, we conduct evaluations with real experiments.

C. Optimizations

The OPPRF-PSI library code as well as some of its dependencies make use of x86 instruction set extensions like Streaming SIMD (Single Instruction Multiple Data) Extensions (SSE) and its successors as well as common crypto extensions. Modern ARM CPUs widely used in mobile devices have their own 64-128bit SIMD instruction set called NEON. It is available since ARM Architecture, Version 7 (ARMv7) [23]. The x86 and NEON intrinsic functions are different and there is no one-to-one correspondence between them [23]. Nevertheless, projects like sse2neon [24] offer translations from SSE to ARM NEON intrinsics. As we were not able to port parts of the library and protocol source files to use NEON, the intrinsics are disabled for the Android library port.

V. Evaluation

The key goal of this evaluation is to collect data from experiments to find out how well CERTAIN performs. This is mostly a question of efficiency. The metrics used to evaluate the app were runtime, communication as well as CPU usage and energy consumption on a smartphone. The evaluation has to cover different scenarios for parameters like network environment and circuit functionality. The impact of different protocol phases is also of interest.

For the experiments a Lenovo Thinkpad T480s with an Intel Core i5-8250U (4 Cores at 1.60-3.40 GHz) and 16 GB of RAM is used as the server. The client is an OnePlus 5 with Android 9, a Qualcomm Snapdragon 835 Octa-core processor, 6 GB of RAM and a 3,300 mAh battery. The app is compiled with target SDK version 28 as arm64-v8a ABI (Application Binary Interface) to match the requirements of the evaluation hardware.

A. Network Setup

Three different network environments are evaluated using the network emulator NetEm [25] to add additional delay, packet loss and rate limiting on the server's network interface. In the baseline LAN environment server and mobile phone are connected to the same local network via Ethernet and 5 GHz WiFi. It has no packet loss, an RTT of 2.49 ± 0.19 s, 460.1 ± 46.8 MBit/s downstream and 488.87 ± 70.3 MBit/s upstream. The WAN environment setup represents the case that the mobile phone communicates with a remote server via a stable high-bandwidth connection. Here, packet loss is set to 0.01%, RTT is 40.14 ± 0.7 s, downsteam is 17.5 ± 3.8 MBit/s and upstream is 17.9 ± 4.1 MBit/s. The RTT and throughput values are aligned to the test setup of Kolesnikov et al. [26]. The LTE environment simulates the situation where the mobile phone is connected to a mobile network and communicates to a distant server over a heavily asymmetric connection. Packet loss is also set to 0.01%, RTT is 50.61 ± 1.65 s, the connection from server to client has 13.6 ± 2.8 MBit/s while the opposite direction has 3.8 ± 2.0 MBit/s. RTT and throughput values and their standard deviation are measured with the ABY benchmarking tool and iperf3 [27] over at least 20 test runs per data point.

B. Results Data and Communication

As Figure 1 shows, complex risk scoring functionality heavily impacts both runtime and communicated data. Summation of payload provided by the client does not differ from a circuit that does not perform any functionality on top of PSI (see Figure 1, *Analytics*). When the the server also provides payload, runtime sharply increases and the amount of data sent doubles. For the most complex variant of risk scoring following DP-3T or GAEN, multiplications need to be added. We can see that while runtime only rises slightly from *ABM*/*ABMT* to *ABS*/*ABST*, the amount of data to be sent increases drastically. For the functionalities *ABM*/*ABMT* the client is also required to send and receive more data. As we see in Figure 2 runtime for this circuit is heavily impacted by the asymmetric LTE connection. Revealing results only when a certain threshold value is surpassed has negligible influence on communication and runtime for any of the circuits.

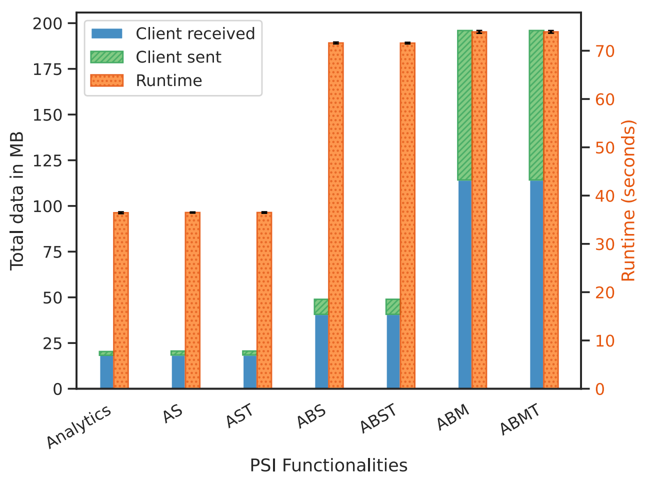

Figure 1. Time and data consumption for different risk scoring functions for calculating the risk score for one day. Runtimes and communication data for all different circuits with $n_1 = 2^{10}$ and $n_2 = 2^{19}$ and 2 bit payloads. Runtime means measured with 20 runs each in the LAN setting. Error bars show the standard deviation.

Standard deviations for the runtime are small within the LAN network. This stability in measurements is caused by a small RTT, high throughput and almost non-existent packet loss rate in the LAN network. This changes for the WAN and LTE networks as visualized by the larger error bars. Additional experiments have shown that not emulating additional packet loss does result in a small standard deviation even for the LTE environment. In general, the runtime increases from LAN to WAN to LTE. The small differences of less than 8 s between the networks do not change for the first two circuits. This shows that including payload from only the client has no impact on runtime across all networks. Once payload from the server is included, the differences between the networks get slightly bigger (10–20 s). We also conducted experiments with different server set sizes $n_2 = 2^x$ for $x \in \{10, \ldots, 21\}$ and $n_1 = 2^{10}$ in the LAN. Both duration as well as the amount of transmitted data grow exponentially.

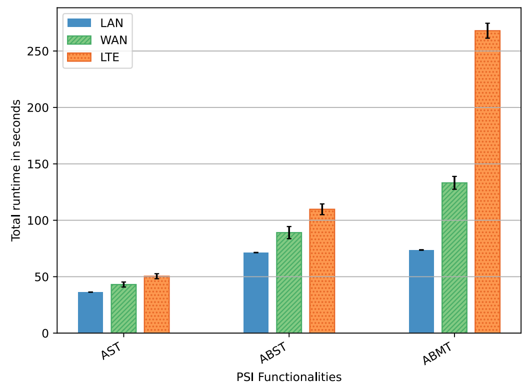

Figure 2. Duration of different functionalities for risk scoring in different networks.

C. Results Energy and CPU Usage

Energy consumption is measured using the Battery Historian tool from Google [28], which is available since Android 5 (Lollipop). With this tool CPU time and estimated battery consumption can be tracked for an app. The means and standard deviations are displayed in Table I.

Table I
MEANS AND STANDARD DEVIATION FOR ESTIMATED POWER USAGE AND CPU TIME MEASURED OVER 20 RUNS WITH $n_1 = 2^{10}$ AND $n_2 = 2^{19}$.

Circuit	Analytics	AST	ABT	ABMT
Est. Power Usage (%)	0.031 ± 0.002	0.034 ± 0.005	0.071 ± 0.003	0.215 ± 0.005
CPU Time (ms)	3507 ± 54	3442 ± 93	4753 ± 74	7960 ± 93

The estimates for power usage are given as percentage of the battery charge capacity (3,300 mAh) that the app consumed during the execution. The *ABMT* circuit reaches 0.21% of power usage while the others are between 0.03% and 0.07%. The increased circuit complexity causes a three-fold increase of energy consumption.

The Battery Historian tool also provides an estimate for the power use due to CPU usage. This is reported as 0.00% for all runs even though Table I shows that an app execution takes between 3 s to 8 s of CPU time. This value is the sum of the CPU user time and system time. The CPU system time always amounts to less than 30% of the user time. It is unclear whether WiFi energy consumption is accurately accounted for in the estimated power usage for the app. System-level WiFi is responsible for less then 0.03% of energy consumption during app executions.

D. Results for Different Payload Lengths

To evaluate the impact of different payload lengths, experiments were conducted with $\delta \in \{2, 3, 4\}$ bit payload for an ABMT circuit. Changing from 2 bit to 3 bit, requires the client to send an additional 31.5 KB of data and receive additional 31.2 KB. Comparing payload of size 4 bit with the baseline gives additional 65.0 KB send and 64.5 KB received. The increase of runtime is minor with a around 60 ms from 2 bit to 4 bit payload.

VI. Discussion

A. Unbalanced Sets

The original protocol design for OPPRF-PSI was only managing balanced sets. While it was easy to increase server set size without directly increasing the number of bins, the protocol parameters had to be adjusted to uphold the security guarantees and implementation restrictions. The implementation decision of using the Mersenne-prime field and the failure probability for collisions enforced a maximum bit-length of intermediate values. This affected the batch-OPPRF in a way that bigger hint polynomials had to be used. A hint size limit of 1024 related to interpolation performance enforced the batch-OPPRF to split them into more and smaller polynomials. This grows more and more inefficient, both computation and communication-wise. In evaluations of Pinkas et al. for the case of balanced sets the transport of polynomials required less than 3% of the total communication data [12]. This changes drastically in the unbalanced set case. In our measurements, polynomials are responsible for more than 60% of protocol communication for the less advanced circuits. For more complex circuits this number decreases due to the increasing circuit size.

B. Efficiency

To get the total communication data and execution times required for 14-day risk scoring, the OPPRF-PSI evaluation results from Figure 1 have to be multiplied by 14. To decrease runtime, parallelization can be leveraged. Communication for executing OPPRF-PSI 14 times a day can be up to multiple gigabytes and are therefore too high for an DCT system which has to be efficiently scalable. One method to improve efficiency could be reducing the infectious period to 10 days, as used for example by DP-3T. Communication would be decreased by almost 30%, as only 10 OPPRF-PSI would be executed each day.

By handling input-independent communication, e.g. from the circuit setup phase, differently and reducing the infectious period, the communication values are less impractical, but still reach above 1 GB of data. As the more complex circuits are a main contributor to communication, OPPRF-PSI can be used efficiently for DCT at least in a scenario with 5000 uploads per day, if advanced risk scoring functionality is not applied. Another optimization option would be to outsource the client's circuit computation to a set of untrusted independent servers as described by Duong et al. [10]. As these outsourcing servers and the HA's server would be in a LAN or WAN setting performance would greatly improve. Also the amount of data communicated by the client would decrease to $\mathcal{O}(n_1)$ independent of the computed risk scoring function.

C. Risk Scoring Payloads

To produce risk scoring following GAEN v2, the client can pre-compute a risk value according to the duration-at-attenuation for each EP following Equation 1 and input this value using only a few payload bits. As we have seen in the experiments, increasing payload size has only little impact on the runtime. For each additional bit of payload length more data has to be communicated, resulting in a constant overhead of about 62.7-64.75 KB per bit up for relevant sizes assuming linear growth. In case of 16-bit payloads, another 906,5 KB of data would have to be communicated between client and server. For an even more fine-grained risk scoring approach EP specific attenuation values could be used as payload.

D. Privacy and Security

1) OPPRF-PSI: The OPPRF-PSI protocol protects against a semi-honest attacker as described in Section III-A. This means that if the attacker follows the protocol, no private information is leaked. A malicious attacker on the other hand might try to deviate from the protocol to either learn private information or break the functionality of the protocol. To defend against a malicious attacker, all parts of the risk scoring functionality have to be secured. In the publication of OPPRF-PSI [12] no maliciously-secure design was proposed. But the authors noted that modern circuit-PSI protocols based on cuckoo hashing have to rely on the correct hashing of the parties. It is inherently hard to extend protocols based on cuckoo hashing to obtain security against malicious adversaries. This is because the placement of items depends on the exact composition of the input set, and therefore a malicious party might learn the placement used by the other party [29]. Since OPPRF-PSI only applies cuckoo hashing on the client side, this risk only exists in case of a malicious server. In [30] PaXoS is used, a new data structure for malicious-secure Cuckoo hashing to avoid information leakage. This data structure is not applicable to OPPRF-PSI. The simple hashing into bins could be made more secure with an Encode-Commit scheme as proposed in [29].

Several techniques exist to secure MPC protocols for the circuit phase against a malicious adversary. Among those techniques are cut-and-choose, committed OT, authenticated secret sharing, zero knowledge proofs and authenticated garbling [31]. All of these measures heavily impact performance.

The only circuit-based PSI protocol that can be easily secured against malicious adversaries is the SCS protocol [32] by using an additional circuit of size $O(n)$ [12].

As we see, a fully malicious-secure OPPRF-PSI is hard or even impossible to construct due to the use of cuckoo hashing and absence of malicious-secure sub-protocols. Switching out some sub-protocols with their malicious-secure variants induces heavy performance penalties. Neither a semi-honest, nor a malicious-secure OPPRF-PSI protocol is secure against crafted input sets of either party. Such simple attacks can be made infeasible using mitigation tactics such as rate limiting, threshold circuit functionalities or even device attestation.

2) CERTAIN: Let us now take a look at CERTAIN as a whole. An *evil user* might be interested to find out which of the collected EPs belong to infected people. In the semi-honest setting, no EPs of the infected people and no information about the time of encounter are leaked by CERTAIN. This is due to the fact that only aggregated risk scores are returned to the client. Additionally, inputs of client and server are protected from the other side by MPC. Combined, this mitigates deanonymisation attacks based on the time of encounter (which are possible for example in GAEN). To gain access to the server's set of EPs the evil user has to act as malicious adversary during PSI (see Section VI-D1). To ensure that users do not behave like a malicious adversary, app attestation mechanism can be used which prove integrity of the application. To stop evil users from repeatedly querying the server with different subsets of their data, a threshold function, which only releases the true risk value if it exceeds a certain level, is also applied. This measure can be combined with limiting a user's number of queries per day. An evil user might try to issue false warnings. To defend against this attack a token mechanism of DP-3T can be employed to ensure users can only provide data to the server for PSI if they are verified as infected [18].

An *evil HA* that is semi-honest does not learn if a querying user is at risk or who they interacted with due to the fact that an MPC protocol is used. To ensure that the HA does not behave maliciously, trusted computing and remote attestation mechanisms can be used. This allows audits to ensure that the server runs the correct software.

An *eavesdropper* listening on network traffic can learn who uploaded data to the servers of the HA and identify diagnosed users. As mitigation, data collection can be either distributed to testing centers or all users randomly initiate dummy data uploads to the server. Another way the eavesdropper can learn who is infected or who has recently been fully vaccinated is when a user stops computing risk scores daily. As a result, even if a user is diagnosed they should continue to query the server. Attacks on BLE such as relay or replay of EPs are the same as for GAEN and other BLE-based DCT approaches (see [2]).

VII. Conclusion

In this work we present our approach CERTAIN for digital contact tracing based on OPPRF-PSI. We implemented support for server and client-side payload which is then used to realize complex functions for risk scoring similar to the mechanisms of DP-3T and GAEN. We evaluated the protocol for unbalanced set sizes on an Android device. The selected parameters represent a realistic pandemic situation. It was shown that while computation load and energy usage on the client devices is small the amount of data to be communicated has a heavy impact especially on asymmetric networks. Outsourcing client computation to a set of untrusted as proposed by Duong et al. can solve this issue.

References

[1] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *EUROCRYPT 2019*, ser. Lecture Notes in Computer Science, vol. 11478. Springer, 2019, pp. 122–153.

[2] L. Reichert, S. Brack, and B. Scheuermann, "A Survey of Automatic Contact Tracing Approaches Using Bluetooth Low Energy," *ACM Trans. Comput. Heal.*, vol. 2, no. 2, pp. 18:1–18:33, 2021.

[3] C. Wymant *et al.*, "The epidemiological impact of the NHS COVID-19 App," *Nature*, pp. 1–8, 2021.

[4] Google and Apple, "Privacy-preserving contact tracing," 2020, accessed: 24. June 2021. [Online]. Available: www.apple.com/covid19/contacttracing

[5] Corona-Warn-App open-source project, "Kennzahlen zur Corona Warn App (Stand 10. Juni 2021) ," 2021, accessed: 14. June 2021. [Online]. Available: coronawarn.app/assets/documents/2021-06-10-cwa-daten-fakten.pdf

[6] N. P. Smart, *Cryptography Made Simple*, ser. Information Security and Cryptography. Switzerland: Springer, 2016.

[7] A. Berke, M. A. Bakker, P. Vepakomma, R. Raskar, K. Larson, and A. S. Pentland, "Assessing disease exposure risk with location histories and protecting privacy: A cryptographic approach in response to a global pandemic," *CoRR*, vol. abs/2003.14412, 2020.

[8] D. Demirag and E. Ayday, "Tracking and controlling the spread of a virus in a privacy-preserving way," *CoRR*, vol. abs/2003.13073, 2020.

[9] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song, "Epione: Lightweight contact tracing with strong privacy," *CoRR*, vol. abs/2004.13293, 2020.

[10] T. Duong, D. H. Phan, and N. Trieu, "Catalic: Delegated PSI cardinality with applications to contact tracing," in *ASIACRYPT 2020*, ser. Lecture Notes in Computer Science, vol. 12493. Springer, 2020, pp. 870–899.

[11] L. Reichert, S. Brack, and B. Scheuermann, "Privacy-preserving contact tracing of COVID-19 patients," Poster Session at the 41st IEEE Symposium on Security and Privacy, 2020.

[12] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient Circuit-Based PSI with Linear Communication," in *EUROCRYPT 2019*, vol. 11478. Springer, 2019, pp. 122–153.

[13] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private Set Intersection Using Permutation-based Hashing," in *USENIX Security 2015*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 515–530.

[14] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, "Efficient Circuit-Based PSI via Cuckoo Hashing," in *EUROCRYPT 2018*, vol. 10822. Springer, 2018, pp. 125–157.

[15] M. Ciampi and C. Orlandi, "Combining Private Set-Intersection with Secure Two-Party Computation," in *SCN 2018*, ser. Lecture Notes in Computer Science, D. Catalano and R. D. Prisco, Eds., vol. 11035. Springer, 2018, pp. 464–482.

[16] R. Pagh and F. F. Rodler, "Cuckoo Hashing," in *ESA Symposium 2001*, ser. Lecture Notes in Computer Science, vol. 2161. Springer, 2001, pp. 121–133.

[17] Apple Inc., "ENExposureConfiguration," 2020, accessed: 14. June 2021. [Online]. Available: developer.apple.com/documentation/exposurenotification/enexposureconfiguration

[18] C. Troncoso *et al.*, "Decentralized Privacy-Preserving Proximity Tracing," 2020, accessed: 15. June 2021. [Online]. Available: www.github.com/DP-3T/documents

[19] Encryptogroup, "OPPRF-PSI," ENCRYPTO TU Darmstadt, Nov. 2020, 15. June 2021. [Online]. Available: https://github.com/encryptogroup/OPPRF-PSI

[20] D. Demmler, T. Schneider, and M. Zohner, "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation." Internet Society, 2015.

[21] Android Developers, "Android NDK," accessed: 15. June 2021. [Online]. Available: developer.android.com/ndk

[22] Robert Koch Institute, "Corona Virus Disease 2019 - Daily Situation Report of the Robert Koch Institute," 2020, accessed: 14. June 2021. [Online]. Available: rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Dez_2020/2020-12-22-en.pdf

[23] Intel Corporation, "From ARM NEON* to Intel® SSE," accessed: 15. June 2021. [Online]. Available: intel.com/content/www/us/en/develop/articles/from-arm-neon-to-intel-sse-the-automatic-porting-solution-tips-and-tricks.html

[24] Distributed Ledger Technology Collaboration, "DLTcollab/sse2neon," Dec. 2020, accessed: 15. June 2021. [Online]. Available: github.com/DLTcollab/sse2neon

[25] S. Hemminger, F. Ludovici, and H. P. Pfeifer, "Tc-netem(8) - Linux manual page," accessed: 24. June 2021. [Online]. Available: man7.org/linux/man-pages/man8/tc-netem.8.html

[26] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient Batched Oblivious PRF with Applications to Private Set Intersection," in *ACM SIGSAC 2016*. ACM, 2016, pp. 818–829.

[27] The iPerf Project, "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool," accessed: 15. June 2021. [Online]. Available: iperf.fr

[28] Google Developers, "Analyze power use with Battery Historian," 2021, accessed: 15. June 2021. [Online]. Available: developer.android.com/topic/performance/power/battery-historian

[29] P. Rindal and M. Rosulek, "Malicious-Secure Private Set Intersection via Dual Execution," in *CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 1229–1242.

[30] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from PaXoS: Fast, Malicious Private Set Intersection," in *EUROCRYPT 2020*, ser. Lecture Notes in Computer Science, vol. 12106. Springer, 2020, pp. 739–767.

[31] D. Evans, V. Kolesnikov, and M. Rosulek, "A Pragmatic Introduction to Secure Multi-Party Computation," *Found Trends Priv Secur*, vol. 2, no. 2-3, pp. 70–246, 2018.

[32] Y. Huang, D. Evans, and J. Katz, "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?" in *NDSS 2012*. The Internet Society, 2012.