DIMY: Enabling Privacy-preserving Contact Tracing

Nadeem Ahmed, Regio A. Michelin, Wanli Xue, *Member, IEEE,*
Guntur Dharma Putra, *Student Member, IEEE,* Sushmita Ruj, *Senior Member, IEEE,*
Salil S. Kanhere, *Senior Member, IEEE,* and Sanjay Jha, *Senior Member, IEEE,*

Abstract—The infection rate of COVID-19 and lack of an approved vaccine has forced governments and health authorities to adopt lockdowns, increased testing, and contact tracing to reduce the spread of the virus. Digital contact tracing has become a supplement to the traditional manual contact tracing process. However, although there have been a number of digital contact tracing apps proposed and deployed, these have not been widely adopted owing to apprehensions surrounding privacy and security. In this paper, we propose a blockchain-based privacy-preserving contact tracing protocol, ”Did I Meet You” (DIMY), that provides full-lifecycle data privacy protection on the devices themselves as well as on the back-end servers, to address most of the privacy concerns associated with existing protocols. We have employed Bloom filters to provide efficient privacy-preserving storage, and have used the Diffie-Hellman key exchange for secret sharing among the participants. We show that DIMY provides resilience against many well known attacks while introducing negligible overheads. DIMY’s footprint on the storage space of clients’ devices and back-end servers is also significantly lower than other similar state of the art apps.

Index Terms—COVID-19, Contact Tracing, Bloom Filter, Blockchain, Privacy, Security.

1 Introduction

The outbreak of the COVID-19 pandemic has changed many aspects of everyone’s way of life. One of the characteristics of COVID-19 is its airborne transmission, which makes it highly contagious. Moreover, a person infected with COVID-19 can be asymptomatic, thus spreading the virus without showing any symptoms. Anyone who comes into close contact[1] with an infected person is at a high risk of contracting the coronavirus. The lack of an approved vaccine has led governments to enforce lockdowns, quarantines and to recommend social distancing, aiming to prevent the spread of COVID-19. However, despite these precautionary measures, the rate of spread of COVID-19 is putting the public health systems of many countries under strain.

Contact tracing is an established technique that has proven successful in dealing with other outbreaks such as Ebola and SARS. Contact tracing aims to establish the close-contacts of an infected person so that they may be tested/isolated to break the chain of infection. Traditionally, the contact tracing process is performed manually in a *reactive* manner, triggered when a person tests positive to the virus. This is achieved by conducting a face-to-face interview to establish contacts made by the person while infectious[2]. This approach, although useful, has some limitations: *i*) It requires a large trained workforce to cope with the caseload. *ii*) It is hard for people to remember everyone they have met while infected in the last 2-3 weeks. *iii*) A person may have met people that are strangers. Proactive contact tracing [1], [2], [3], [4] has recently been proposed to mitigate these issues by maintaining a record of all close contacts made by a person and utilising these records if that person tests positive.

One way of implementing proactive contact tracing is to mandate record-keeping of people’s attendance at public venues such as offices, restaurants, etc. This can be done manually , for example, through QR codes that direct attendees to register their details. However, this increases the risk to individuals’ data privacy and allows for possible tracking of the user’s behaviour. A more popular option is to employ smartphone-based digital contact tracing apps that can exchange Bluetooth Low Energy (BLE) messages with each other to record this contact. The digital contact tracing app is typically composed of two main entities, the smartphones acting as clients and a back-end server. In this model, the smartphones of two individuals with tracing apps installed would exchange some random identification code (this identification code does not reveal any sensitive information about their actual identities) when they are in close proximity. The back-end is typically maintained by health organisations (or the government), and once a person is diagnosed with COVID-19, they can opt to share the local list of contacts stored on their smartphone with the back-end server to identify at-risk users.

The popularity of digital contact tracing apps can be

- *N. Ahmed, R. Michelin, W. Xue, G. Putra, S. Kanhere and S. Jha are with the Cyber Security Cooperative Research Centre (CSCRC) - Australia and University of New South Wales (UNSW) - Sydney, Australia. Corresponding author: nadeem.ahmed@cybersecuritycrc.org.au*
- *S. Ruj is with CSIRO, Data61 - Sydney, Australia.*

Manuscript received November 18, 2020; revised January 26, 2021.

1. According to the Centres for Disease Control and Prevention (CDC, https://www.cdc.gov/coronavirus/2019-ncov/downloads/2019-ncov-factsheet.pdf), close contact with an infected person is defined as a contact within a range of 6 feet for approximately 15 minutes.

2. The infectious period for a COVID-19 positive case is considered as 2-3 weeks including the asymptomatic period.

gauged by the fact that more than 45 such apps have been proposed or are being used in by different countries [5]. These COVID-19 digital contact tracing apps are based on different architectures distinguishable in several aspects, including anonymous ID generation and exchange, risk analysis and notifications, etc. For details, readers are referred to a recent survey on digital contact tracing apps by Ahmed *et al.* [6], in which the architectures are classified as centralised, decentralised and hybrid, according to the distribution of key functionalities among the clients and the back-end server. However, recent security and privacy analyses of these apps has revealed several risks and issues [6], [7], [8]. These apps operate on different trust models. Apps based on the centralised architecture (such as [9], [10], etc.) generally collect sensitive data at the server, that are assumed to be trusted, and only provide privacy protection against malicious users. This trust model makes these apps vulnerable to server-side breaches and malicious actions by the server. On the other hand, apps based on decentralised and hybrid architectures assume an honest-but-curious server model whereby the server will try to harvest sensitive information, if available. Apps such as [11] and [2] that are based on the decentralised architecture share the anonymous identifiers of the positive cases with all users for matching, making these apps vulnerable to linkage attacks, whereby malicious users can discover the real identities of persons diagnosed with COVID-19 [7]. Apps based on hybrid systems perform the risk analysis and notification process at the server instead of revealing the anonymous IDs of positive cases to other users for matching, as proposed in the decentralised architecture. However, these apps suffer from high communication and processing costs. For example, the DESIRE protocol [12] uses three BLE messages to advertise a single anonymous ID from a device [11], while the ContraCorona app [13] employs three non-colluding servers (submission, matching and notification servers) to manage the contact tracing process.

In this paper, we propose a new privacy-preserving digital contact tracing protocol called ”Did I Meet You” (DIMY) that can be classified in the hybrid category. We take a holistic view of the privacy and security requirements for digital contact tracing and employ techniques to address most of the concerns associated with existing contact tracing protocols.

We make the following specific contributions:

- DIMY has been designed to provide full life cycle data privacy protection that prevents contact tracing data from being used arbitrarily. This is achieved by using the Diffie-Hellman key exchange and a secret sharing mechanism, to establish a secret contact representation between user devices over an inherently insecure BLE broadcast channel. We also employ Bloom Filters for storing close contact information both at the individual device level as well as the back-end. Additionally, information from multiple close contacts are stored in a single fixed-size Bloom filter. This contact information is deleted from the user’s device once it is encoded in the Bloom filter, which serves two important purposes: (i) It prevents information leakage not only at the client level (for example as a result of device theft or coercion attacks), but also from authorities operating the back-end and governments that can obtain subpoenas to access information stored on the back-end. (ii) It considerably reduces client device and back-end storage requirements.
- As opposed to traditional apps that employ centralised servers at the back-end, we have improved the scalability and security of our proposed solution by employing a blockchain-based back-end design in the ecosystem. This provides transparency and trust on back-end operations besides ensuring the integrity of data uploads from positively identified cases that are appended as blockchain transactions. We also evaluate the performance of our implementation based on Hyperledger and show that DIMY provides low latency and resource consumption while supporting high throughput under moderate loads.
- We consider a comprehensive threat model and provide safeguards against several types of adversaries including malicious users, back-end (admin and developers) and government (discussed in detail in Section 6.1). We also provide a comprehensive security and privacy analysis of our proposed solution and show that DIMY provides resilience against common attacks such as linkage, enumeration, social graph construction and replay (discussed in detail in Section 6.4).

This paper is organised as follows. We discuss related work in Section 2. Section 3 introduces the background information necessary to understand the building blocks of our proposed solution. We detail the design of our DIMY protocol in Section 4. In Section 5, we compare the salient features of DIMY with other existing protocols. Section 6 provides a security and privacy analysis of DIMY protocol, while Section 7 details the performance analysis of our proposed solution. Section 8 concludes this paper.

2 Related work

There have been a number of digital contact tracing apps proposed, developed and deployed to aid in identifying exposure from infected individuals. Most of the apps are based on BLE message exchanges, while some of the proposed apps also employ location tracking based on GPS. The MIT Technology Review summarised the salient features of 47 such apps [5]. These apps follow different approaches for development and addressing multiple aspects in terms of privacy, security, performance, and reliability etc. We follow the classification criteria discussed by Ahmed et al., in [6] to classify tracing apps in centralised, decentralised and hybrid categories according to the underlying application architecture and the functionalities delegated to client devices and the server.

2.1 Centralised approach

BlueTrace protocol [1] is one of the first proposed digital contact tracing protocols that is based on a centralised architecture. This protocol was employed to develop the Singaporean TraceTogether [9] and the Australian Covid-Safe [10] apps. Another protocol named ROBERT [14] was proposed by Inria and Fraunhofer AIESEC that is also based on the centralised architecture.

In the centralised architecture, a central server is responsible for handling major components of the digital contact tracing process such as ID generation, risk analysis and notification, etc. Typically in this architecture, a user enrols with the central authority, which periodically (typically every 10-15 min) generates a unique temporary ID for each client. This temporary ID is sent to the user and is used in his/her advertisement message. The user records the received temporary IDs locally when in the proximity of other contacts running the same app. If a user gets diagnosed with COVID-19, a health officer authorises the user to upload (share) the list of all captured IDs to the centralised server for risk analysis and notification of close contacts.

The central server in the BlueTrace protocol can access the personally identifiable information collected at the registration stage and map each client to their temporary IDs. This raises issues with privacy as this sensitive data can be used for other purposes besides digital contact tracing. In contrast to BlueTrace, ROBERT protocol does not store any user identifiable information on the server. Temporary IDs are still created at the server without been linked with the devices used by the clients. ROBERTS's notification process requires the uploading of IDs used by a device to check whether they have come in contact with a COVID-19 positive case or not. This is in contrast with BlueTrace where the server can identify at-risk users and contact them proactively.

ROBERT protocol, however, similar to other protocols based on centralised architectures, has a high potential to *function creep*, in which it can be re-purposed into a mass surveillance system [11]. Another potential issue associated with centralised architectures is the construction of partial social graphs (discussed in detail in Section 6) that enable linkability of infected cases and their contacts. A server breach can also result in the loss of sensitive data stored at the server.

2.2 Decentralised approach

The decentralised architecture differs from the centralised version by pushing some functionalities to the user's devices. There is still a server involved, however, the role played by the server is more in terms of orchestrating the clients. This approach claims to improve user privacy by generating temporary IDs in the user's devices. Additionally, exposure risk processing is also performed at the device level.

Generally, devices generate random seeds for forming their temporary IDs. These IDs are exchanged with other users who they come in contact with. Once a user is diagnosed positive with COVID-19, all seeds used by the device (some of the apps upload IDs instead of seeds) are uploaded to the server. Any user who wishes to check whether they are at-risk can download the seeds (or IDs) uploaded by the diagnosed users. The device can then perform matching locally, with the user notified of the result. The server is neither involved in the ID generation nor the at-risk analysis and notification process.

There are a number of protocols that follow the decentralised architecture such as DP-3T [11], PACT-East Coast [2], Google Apple Encounter Notification (GEAN) [4], [3] and TCN [15]. They have minor differences in the implementation of sub-components with the basic design following the general functionality described in this section.

Apps based on decentralised architectures provide enhanced privacy protection against server-based attacks as devices generate their own anonymous IDs. However, decentralised apps are known to be vulnerable to linkability attacks, whereby a user who has received the IDs generated by an infected user is able to link the IDs with the real user's identity [7]. These apps are also subject to enumeration attacks, enabling the counting of all positive cases by each user.

2.3 Hybrid approach

Hybrid architectures balance the tasks between the client and the server. The server is responsible for performing the risk analysis and notification process, while the client manages the generation of temporary IDs. Desire [12] is one of the example protocols that follow the hybrid architecture. Devices using the hybrid protocol cryptographically generate and exchange IDs with other devices. A contact between two devices is represented by a unique encounter ID, which the app generates by combining own and the received temporal IDs. A user who tests positive can optionally upload the generated encounter IDs to the server. Any user who wants to check the risk of exposure sends their collected encounter IDs to the server for matching. The server performs risk analysis and notifies any user who is deemed to be at risk.

The Desire protocol uses 256-bit IDs that are broadcast for generating encounter IDs (or tokens) using the Diffie-Hellman key exchange. This design choice requires at least three BLE message exchanges (Advertisement, Scan_Request and Scan_Response) resulting in an increase in energy consumption for devices [16].

2.4 Discussion

We have listed the modalities involved in the design of the three types of architectures commonly used for digital contact tracing. We have also highlighted some common issues related to privacy and security that are associated with apps based on these architectures. Our proposed solution, DIMY, can be broadly classified as a hybrid architecture in that we generate the IDs on the devices, and perform risk-analysis and notification tasks at the server.

For DIMY, we utilise Bloom filters to encode the encounter ID generated by the devices and to store the encounters at the back-end. The DP-3T protocol also suggests the use of Cuckoo filters, but in a different context. In their proposal, the server has access to all the seeds uploaded by users who have tested positive, and hence can generate the IDs used by positive cases. They proposed encoding these IDs in a Cuckoo filter to hide them from other users who are performing local risk-analysis. In comparison, our use of Bloom filters provides better privacy protection as these are used to hide the encounter information both at the device level as well as at the back-end.

We also employ blockchain technology to manage the back-end processing. BeepTrace [17] is another framework that has proposed the use of two blockchains: 'tracing chain' to manage tracing/contact matching with anonymised user data, and the other 'notification chain' to manage notifications at the back-end. In contrast, we propose using a single blockchain (Hyperledger Fabric) and enhancing its privacy protection further by using Bloom filter-encoded data storage. Additionally, we rely on the smart contract functionality to perform the exposure risk-analysis and matching in a privacy-preserving manner. Lv et. al. [18] propose *Bychain*, a new blockchain that stores contact information securely. Bychain protects user identities using Zero-Knowledge protocols. Though the contact information is stored on the blockchain, the authors do not discuss how data is retrieved and used when individuals test positive for COVID-19. In addition, the proposed protocols rely on support from GPS equipped provers and witnesses for recording contact information using LTE, WiFi and BLE. Our proposed design, in contrast, is an end-to-end BLE based solution for contact tracing relying on widely available BLE modules on smartphones.

We also note that in the context of digital contact tracing, use of cryptographically generated IDs and the Diffie-Hellman key exchange mechanism has been proposed in [13], [19], [12]. Similarly, the k-out-of-n secret sharing mechanism for ID distribution has been proposed as extension to the standard protocols in [11], [13]. In our proposed protocol, the secret sharing mechanism is coupled with the Diffie-Hellman key exchange. We integrate these security and privacy-preserving techniques with efficient set membership using Bloom filters and additionally employ blockchain technology at the back-end. Table 1 highlights the key technologies used in DIMY and places our proposal in the context of existing protocols.

TABLE 1
Comparison of key technologies (C=Centralised D=Decentralised H=Hybrid). A $\star$ denotes an extension to the base protocol.

	Key Technologies				
Protocols	DH	Secret Sharing	BF	BC	Architecture
BlueTrace [1]	×	×	×	×	C
CovidSafe [10]	×	×	×	×	C
ROBERT [14]	×	×	×	×	C
DP-3T [11]	×	$\star$	✓	×	D
PACT-East Coast [2]	×	×	×	×	D
GAEN [3], [4]	×	×	×	×	D
Desire [12]	✓	×	×	×	H
Contra Corona [13]	✓	$\star$	×	×	H
BeepTrace [17]	×	×	×	✓	H
ByChain [18]	×	×	×	✓	H
DIMY	✓	✓	✓	✓	H

3 Background Information

In this section, we introduce key technologies which form the building blocks of our proposed solution, including Diffie Hellman key exchange, Shamir secret sharing, Bloom filters, and blockchain.

3.1 Diffie Hellman Key Exchange

Diffie-Hellman [20] is a public key distribution system that addresses the issue of secret key distribution over an insecure channel. It enables two users to communicate with each other in order to arrive at a common symmetric secret key that can be used for encrypting/decrypting their future communications. This secret key is computed in such a manner that an eavesdropper cannot reconstruct the shared secret key, in a computationally feasible context, even if they have heard all the messages exchanged.

This key distribution mechanism is based on the discrete logarithm problem. Let G be a multiplicative group of prime order n. Let $g \in G$ be a generator. Party A chooses $r_1 \in Z_p$, computes g^{r_1} and sends to party B. B chooses $r_2 \in Z_p$, computes g^{r_2} and sends to party A. On receiving g^{r_2}, A computes $(g^{r_2})^{r_1} = g^{r_1 r_2}$, similarly, on receiving g^{r_1}, B computes $(g^{r_1})^{r_2} = g^{r_1 r_2}$. Due to the hardness of the discrete logarithm problem, an adversary cannot compute r_1, given g^{r_1}. Hence, it cannot construct the common key. In our contact tracing protocol, G is an elliptic curve group.

3.2 Shamir Secret Sharing

In our proposed protocol, we use a secret sharing scheme [21] to provide information privacy and secure communication between the devices participating in contact sharing. The basic idea revolves around making shares of a secret that can be securely distributed over many devices by a threshold secret sharing mechanism.

A secret sharing scheme consists of two phases, called *sharing* and *reconstruction*. In a k-out-of-n secret sharing scheme (also referred to as (k, n)-secret sharing scheme), there is a unique player called the dealer who wants to share parts of secret S among n players, $P_1, P_2, ..., P_n$. The dealer creates n shares of the secret S $(S_1, S_2, ..., S_n)$ and sends every player a share (say S_i to player P_i) of the secret S in a way that any group of k or more players can reconstruct the secret. All shares are necessary for the reconstruction of the secret if we keep $k = n$.

A k-out-of-n secret sharing scheme, in general, has to satisfy the following two properties:

1) Recoverability: The secret can be reconstructed given any k shares.
2) Secrecy: No information can be known about the secret given any number of shares $< k$.

A dealer is assumed as honest in standard secret sharing. However, additional information or multiple communication rounds are required to verify the consistency of shares held by various parties leading to the notion of a verifiable secret sharing scheme.

3.3 Bloom Filter

We employ Bloom Filter (BF)-based storage for logging contact information on the devices and at the back-end blockchain. A Bloom filter (BF) [22] is a probabilistic data structure used to represent set membership. It supports an efficient mechanism for set membership queries. When queried, the BF will return true (with a false positive) if the queried data exists in the filter. A BF is implemented as a bit array $BF[i]$, i $\in (1, n)$, of n bits accessed via h independent hash functions $H_1(x)...H_h(x)$, each of which maps an element x in a set S of m elements to one of the l bits within the bit array. Querying the presence/membership of an element x in the set using a BF requires checking $\bigwedge_{j=1}^{h} BF[H_j(x)] = 1$ (i.e., $\bigwedge_{j=1}^{h} BF[H_j(x)]$ returns 1 only if all h corresponding bits are set to 1).

In BFs, false-positives (FP) are possible, but false-negatives (FN) are not. An FP $\psi(m, k, n)$ is the probability that a membership test performed for an element x not stored in $BF(S)$ returns 1, in which the parameter m specifies the size of the bit array (Bloom filter length), k specifies the number of hash functions, and n is the cardinality of the stored set. Even if an exact expression for $\psi(m,k,n)$ is available [23], virtually all work in the field relies on a simple, but tight, approximation:

$$\psi(m,k,n) = \Big(1-(1-\frac{1}{m})^{kn}\Big)^k \approx \Big(1-e^{-kn/m}\Big)^k \quad (1)$$

Simply, an FP is due to the collision of two different elements being mapped to the same bit position.

3.4 Blockchain

Blockchain technology was initially introduced in 2008 to maintain a public ledger of Bitcoin transactions [24]. This technology allows network participants to create chronologically sequential immutable blocks ensuring integrity, trust and transparency. Blockchain technology has since been applied in many different applications, from cryptocurrencies [24], [25] to IoT [26]. It enables the creation of solutions that do not rely on a central authority; rather, the chain is spread over several nodes in a distributed manner. It also ensures information integrity by linking the blocks in the chain through a hash function of the previous blocks.

There are three main types of blockchain; public, private and permissioned. The public instances are blockchains that allow any peer to participate in the network. Some examples are Bitcoin [24] and Ethereum [27]. Private blockchains are restricted networks that allow only some nodes to participate while relying on a central authority to manage the nodes. As an example, Ethereum can also run in a private instance, however, while executed privately, there is no connection/interaction with the public instance. In the permissioned blockchain, a group of participants perform the node access control. The main example is Hyperledger, in which organisations are responsible for managing the network. The Hyperledger Fabric [28] blockchain instance supports the deployment of chaincode, a small piece of source code developed and embedded in the blockchain, and it is executed once a node sends a transaction to its address. It uses a consensus based on the Byzantine general's problem, known as RAFT, which defines a leader to conduct an election with the existing nodes connected to a given organisation. This consensus protocol makes the Hyperledger Fabric blockchain a good choice to support our solution, where the organisations are modelled as health authorities (see Section 4). Using this blockchain technology in our solution allows for data integrity, transparency of operations and decentralised data storage.

4 DIMY Protocol Description

We first provide an overview of the DIMY protocol with a detailed description of the building blocks appearing in subsequent sections. Figure 1 shows the overall architecture for our proposed solution. Consistent with other decentralised and hybrid architecture-based contact tracing approaches, devices participating in DIMY periodically generate random ephemeral identifiers. These identifiers are used in the Diffie-Hellman key exchange (Refer to Section 3.1) to establish a secret key that would represent the encounter between two devices that come in contact with each other. For example, Alice generates a random number X_{At} at time t and calculates its ephemeral identifier $EphID_{At} = g^{X_{At}} \in \{0,1\}^{128}$ ($g \in G$ is a generator and G is an elliptic curve group of order p). After generating their $EphID$, devices employ the k-out-of-n secret sharing scheme to produce n secret shares of the $EphIDs$. Devices now broadcast these secret shares, at the rate of one share per minute, through BLE advertisement messages. A device can reconstruct the $EphID$ advertised from another device if it has stayed in the communication range of this device for at least k minutes, enabling it to collect k secret shares of $EphIDs$. Assume that Alice is able to reconstruct the $EphID_{Bt} = g^{Y_{Bt}}$ advertised by Bob where Y_{Bt} is a random number generated by Bob at time t. Alice now computes the secret encounter identifier $EncID_{ABt} = (g^{X_{At}})^{Y_{Bt}}$. Bob also computes the same encounter identifier Enc_{ABt} having received k advertisements from Alice.

A novel aspect of our proposed solution is the use of Bloom filters for storing contact information. Each device maintains a Daily Bloom Filter (DBF) and inserts all the constructed encounter identifiers in the DBF created for that day. The encounter identifier is deleted as soon as it has been inserted in the Bloom filter. Devices maintain DBF on a 21 days rotation basis, identified as the incubation period for COVID-19. DBFs older than 21 days automatically get deleted.

Our solution employs blockchain at the backend. Once a user is diagnosed with COVID-19, they can volunteer to upload their encounter information to the blockchain. Health Authorities (HA) then generate an authorisation access token from the blockchain that is passed on to the device owner. The user's device combines 21 DBF into one Contact Bloom Filter (CBF) and uploads this filter to the blockchain. The blockchain stores the uploaded CBF as a transaction inside a block (in-chain storage) and appends the block to the chain.

A user who wants to check whether they have come in close contact with any user who was diagnosed positive can query the blockchain on a daily basis. A device combines all of the locally stored DBFs (the maximum number is limited to 21) in a single Bloom filter called the Query Bloom Filter (QBF). The QBF is part of the query that gets uploaded to the blockchain. The blockchain matches the QBF with CBF stored as a transaction in the blockchain and returns "matched" or "not matched" as a response. If the response from the blockchain is negative, the device deletes its QBF. Conversely, if the user is found to be at-risk, the QBF is stored separately for further verification by HA in the contact tracing process.

We now explain each component of the proposed solution in more details.

4.1 Close contact representation

In this section, we briefly discuss the notion of encounter representation in the context of contact tracing apps. One simple way to achieve contact representation involves using device IDs. In this scheme, which we refer to as an ID-based scheme, each device is assigned a temporal ID, either by a central authority server or computed locally at the device. The devices advertise and exchange these IDs. The presence

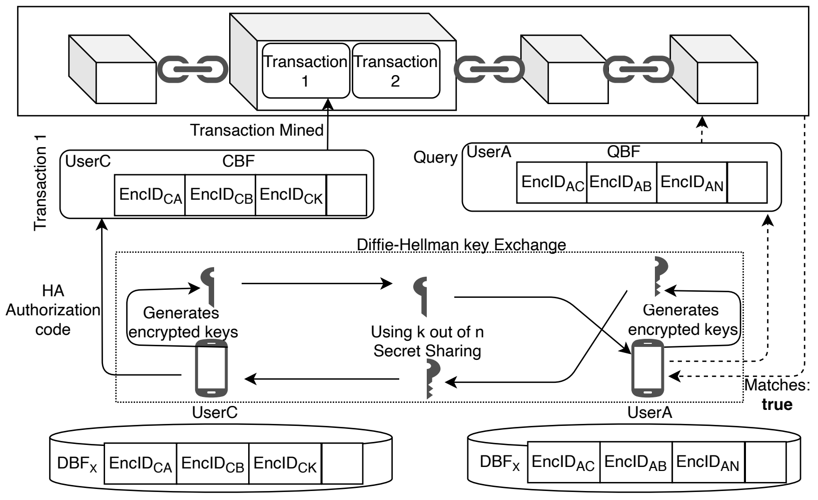

Transaction 1
Transaction 2
Transaction Mined
Transaction 1
UserC
CBF
EncID$_{CA}$ EncID$_{CB}$ EncID$_{CK}$
Query
UserA
QBF
EncID$_{AC}$ EncID$_{AB}$ EncID$_{AN}$
Diffie-Hellman key Exchange
HA Authorization code
Generates encrypted keys
Using k out of n Secret Sharing
Generates encrypted keys
Matches: true
UserC
UserA
DBF$_X$ EncID$_{CA}$ EncID$_{CB}$ EncID$_{CK}$
DBF$_X$ EncID$_{AC}$ EncID$_{AB}$ EncID$_{AN}$

Fig. 1. Basic protocol architecture.

of an ID in the local storage of a device thus represents an encounter with that user (device). In an alternate scheme that we refer to as the shared secret-based scheme, encounters can be represented by a shared secret between two participants. Both participants exchange specific messages to arrive at a shared secret only known to the parties in communication.

In ID-based schemes, all devices in the vicinity of the device A store the same ID advertised by A. In contrast, in the shared secret-based scheme each device pair computes a different shared secret among them. Concretely, if three devices A, B and C meet each other and advertise ID_A, ID_B and ID_C respectively, according to the ID-based scheme, A would store: $\{ID_B, ID_C\}$, B will store: $\{ID_A, ID_C\}$ and C will store:$\{ID_A, ID_B\}$. If these devices are instead using the shared secret-based scheme, they will end up storing secrets as A: $\{S_{AB}, S_{AC}\}$, B: $\{S_{BA}, S_{BC}\}$ and C: $\{S_{CA}, S_{CB}\}$. We have used the shared secret-based representation for recording the encounter between neighbouring devices as these provide more resilience against replay attacks discussed in Section 6. We have employed the Diffie-Hellman key exchange scheme for sharing the secret among communication devices. Figure 2 illustrates the flow of information in the DH scheme used over an insecure BLE communication channel.

4.2 Generating identifiers

This component pertains to generating anonymous device IDs that are used as device advertisements. We consider two common design options: i) Each device generates its own pseudo-anonymous random identifier. This is the approach taken by most of the decentralised and hybrid contact tracing apps such as PACT-East Coast, DP-3T and GAEN. ii) A centralised server generates these identifiers for the registered devices that are then periodically transferred to the devices. This approach is used in apps based on a centralised architecture, such as TraceTogether and COVIDSafe (AU).

In our solution, each device generates their ephemeral IDs, which are valid for 30 minutes. This provides privacy protection against exposing a user's contact details (mapping of IDs to real identities) to the back-end. We have kept the size of $EphID$ as 16 Bytes (128 bits), as BLE advertisement messages are only able to carry a limited payload of data. We note that devices do not directly advertise these $EphIDs$; instead, we use the k-out-of-n secret sharing mechanism (explained in next section).

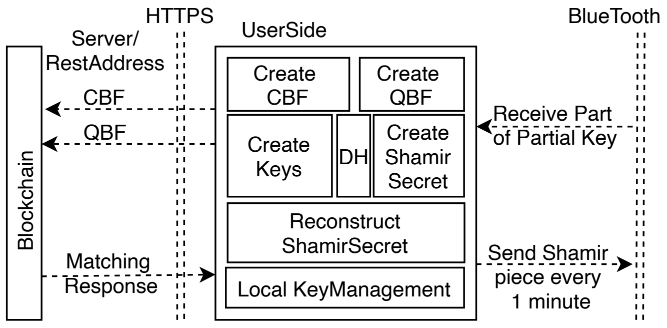
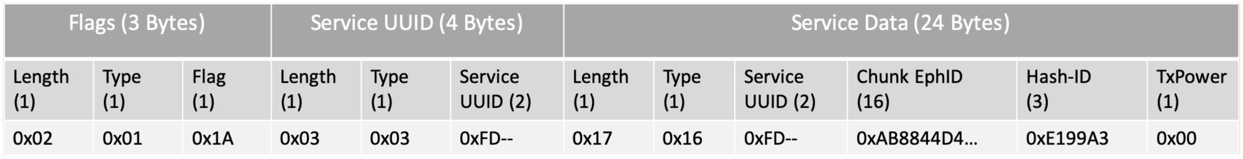


Fig. 2. Information flow in DIMY.

4.3 Advertising and receiving identifiers

Once devices generate the $EphIDs$, the advertisement phase can commence. For this phase, instead of directly advertising the $EphID$, we use a k-out-of-n secret sharing (Shamir's secret sharing) [21] mechanism (explained in Section 3.2). The device calculates n secret part of the $EphID$ and broadcasts each share at the rate of 1 share advertisement per minute. A receiver can reconstruct the $EphID$ if it has successfully received any k out of n shares. For this work, we use the value of k and n as 15 and 30, based on the minimum duration of close contact defined as 15min by CDC.

There are multiple advantages of using this k-out-of-n secret sharing. First, the devices need to be in contact for at least k minutes to receive at least k parts of the secret. Setting k = 15 min automatically takes care of the duration of close contact. Any shorter contacts $< k$ minutes are not registered by the receiver. Additionally, we advertise part of the hash of the $EphID$ in each share. Figure 3 shows the BLE advertisement message format with the 3-Byte hash of the $EphID$ included in the advertisement. This is to prevent the receiver from reconstructing the secret based on shares that are either less than k or using shares advertised by different communicating parties. We note that even if the receiver



Flags (3 Bytes)			Service UUID (4 Bytes)			Service Data (24 Bytes)					
Length (1)	Type (1)	Flag (1)	Length (1)	Type (1)	Service UUID (2)	Length (1)	Type (1)	Service UUID (2)	Chunk EphID (16)	Hash-ID (3)	TxPower (1)
0x02	0x01	0x1A	0x03	0x03	0xFD--	0x17	0x16	0xFD--	0xAB8844D4...	0xE199A3	0x00

Fig. 3. BLE advertisement message format.

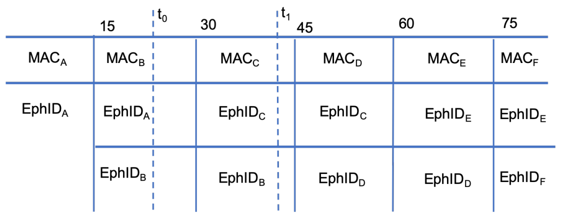


Fig. 4. BLE advertisements with random MACs and $EphIDs$.

uses an incorrect secret value for the encounter, this would never match with any other identifier at the contact tracing stage. In our version, a device will simply discard the shares, without attempting reconstruction, if it has not received at least k advertisement of shares or if the hash values fail the integrity check of the reconstructed $EphID$. Thus, this mechanism increases the complexity for an adversary that is trying to capture the encounter identifiers for malicious use. A computationally bounded adversary, Eve, who is listening for BLE advertisements from Alice and Bob, has to first collect at least k shares of the advertisements from both Alice and Bob. Then, she has to decrypt two random numbers from the reconstructed $EphIDs$ that would take significant time to compute due to discrete logarithm problems associated with the use of decisional Diffie-Hellman [29].

An additional issue associated with using *k-out-of-n* secret sharing is the address carryover mechanism that is due to the rotation of the $EphID$ after each Epoch (30 minutes)[3]. Suppose a receiver device B comes into contact with A when the 10th share of a particular $EphID$ is being broadcast by A at time t_0 and moves away when it has received the 10th share of the next $EphID$ generated by A at time t_1. Device B has thus maintained contact for 20 minutes, however, the logging mechanism would fail to register this contact as it has only received 10 chunks of each $EphID$.

To address this issue, we use the simultaneous advertisement of multiple $EphIDs$ with overlapping intervals as proposed in [13]. A device always broadcast two $EphIDs$, rotating one identifier in such a way that the start of each identifier is staggered by 15 advertisement intervals. In Figure 4, a device is broadcasting two overlapping $EphIDs$. A receiver device which it comes into contact with at time t_0 and leaves at time t_1 is able to register this contact as it has received enough shares of $EphID_B$ while in contact.

As a device is advertising shares of multiple identifiers, we also include the hash of the $EphID$, $Hash(EphID)$ truncated to the first 24 bits (3 Bytes) as the random identifier[4] that is used by the receiver to identify and combine different chunks belonging to the same $EphID$. Once a receiver has collected 15 shares of the same identifier, it reconstructs the identifier and verifies that it has received the correct identifier by computing the hash of the reconstructed ID and comparing the first three bytes with the hash included in the advertisements. Figure 3 shows the message format of the BLE advertisement messages employed in our solution. We have used the ADV_NON_CONN_IND message format for connectionless advertisements for the chunks of EphID.

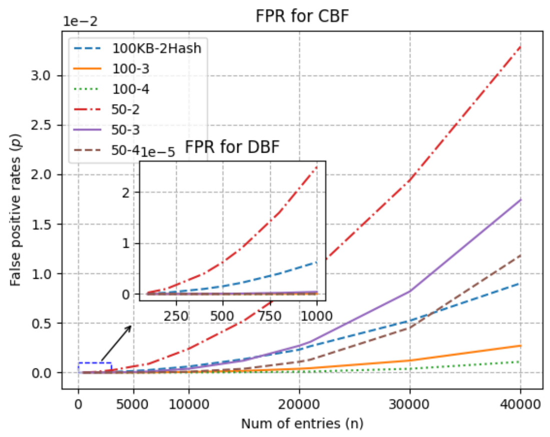


Fig. 5. False positive rate vs number of encounters - DBF and CBF.

4.4 Storing encounter information

After sufficient shares of $EphIDs$ are exchanged with neighbouring devices, each pair of devices can compute a secret symmetric encounter identifier (referred to as $EncID$) only known to them. Each device inserts the encounter identifier in the local DBF. The computed encounter identifier is then deleted after it has been inserted into the DBF. We have used a Bloom filter to preserve the data privacy, reduce the storage requirement and improve the query efficiency, when compared with other normal data storage structures.

Design of Bloom filter: The Bloom filter is a probabilistic structure that can result in false positives. The design of the filter involves various parameters such as the number of entries to be stored in the filter (n), the size of the filter in bits (m), the number of hashes (k) and the false positive rate (p).

Figure 5 shows the false-positive rates for increasing values of encounters n inserted in the DBF and CBF with different values of m and k. Considering DIMY uses the

3. The Epoch is loosely aligned with the randomised MAC address periods that happen at approximately half of the Epoch duration.

4. We take the 128-bit randomly generated $EphID$, pass it through SHA-256 to get a 256-bit hash value, and then truncate it to retain the first 3 bytes.

secret-sharing mechanism (with at least 15 minutes to register a contact), we assume n as 1,000 for DBF as a worst-case representing the maximum number of close contacts per day. As the CBF can hold a maximum of 21 DBFs, the worst-case n for CBF is 21000. The FPR analysis shows that the worst FPR is given by m-k combination of 50 KB-2 Hashes, and the best FPR by 100 KB-4 Hashes. As the hashing is performed at client devices, we take the size of the filter m = 100KB with k as 3 to reduce the computations and battery consumption. The DBF and CBF, in this setting, have FPR of 1 in 19 Million and 1 in 2303, respectively.

4.5 Uploading encounter identifiers to the blockchain

Once a user is diagnosed with COVID-19, they can get an authorisation code from the health authorities to upload their locally stored contact data to the back-end blockchain. Figure 2 shows the information exchange (CBF and QBF to the backend and response from the backend) using a secure channel. The device combines their DBF covering the last 21 days into a single CBF of size 100KB (equal in size to the DBF). The set union function is utilised as the combination process for the DBFs to construct a CBF. For example, all '1'-bit existing information in the DBFs are accumulated into one CBF by performing a bit-wise OR merging [30]. This merged CBF is theoretically equivalent to performing $\sum_{i=1}^{20} DBF_i \cup DBF_{i+1}$ and its false probability is similar to using a standard bloom filter. The CBF is then sent to the backend blockchain for logging as a transaction. The system supports querying by uploading of the QBF (encoded with the DBF from the last 21 days). The user's device uploads this query to the blockchain to check whether someone in close contact has tested positive.

DBF, CBF, and QBF are of the same size of 100KB, serving three distinct purposes: First, it reduces the amount of data transferred to the backend, i.e., instead of 21 100KB sized DBFs, we only use one 100KB CBF or QBF. Second, this aggregation of multiple DBFs into a single CBF and QBF hides the details about the day of encounter to attenuate the privacy threat at the backend. Third, equal-sized CBF and QBF are employed to support efficient Bloom filter matching through set intersection operation at the backend (explained further in Section 4.6).

Design of Blockchain: We use Hyperledger Fabric [31] for the blockchain's implementation, as it provides a modular permissioned blockchain platform, which allows flexibility in modelling the Bloom filter on transactions. The Hyperledger Fabric network is designed to be maintained by a consortium of Health Authorities (HAs) which comprises of stakeholders in the healthcare sector, e.g., relevant government agencies and hospitals. Each HA maintains a set of peer nodes to host the ledgers, execute smart contracts, and maintain a set of orderer nodes for consensus protocol. HAs and their corresponding peers are identifiable by cryptographic primitives that comply with the X.509 standard for public-key certificates.

The HAs interact with the blockchain through multiple smart contracts. We have designed a smart contract that is capable of performing the following functionalities:

- *Issuing access tokens*: Only users who test positive are allowed to upload their CBF to the blockchain. Each user who has tested positive is given a temporary token by the HA that authorises them to access the back-end. The HA transacts with the blockchain to issue the temporary access token by providing corresponding HA credentials to the smart contract. Upon successful credential validation, the smart contract records the token to the blockchain. Note that, this temporary token is only valid for 24 hours.
- *Processing CBF*: The smart contract validates the temporary access token provided by the user who uploads their CBF. Upon successful validation, the smart contract records the CBF permanently to the blockchain and updates the ledger's state.
- *Processing QBF*: The smart contract handles queries from users concerning contacts with positive cases by checking the user's QBF against stored CBF in the ledger. Then, the smart contract returns the matching result, which will either be true or false.

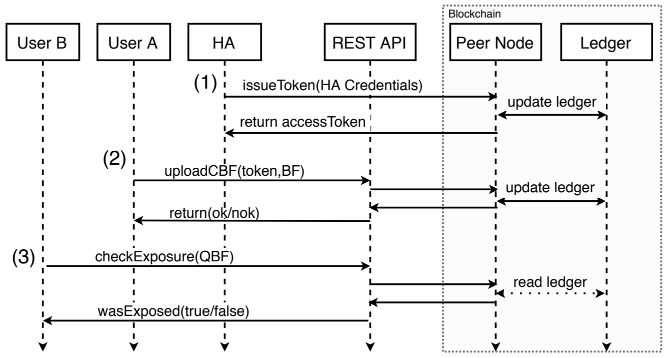


Fig. 6. Uploading to the blockchain.

CBFs stored at the blockchain are managed and queried by the Hyperledger. Given the on-chain data storage capacity of a single transaction is 4MB [32], the Hyperledger can add a minimum of 1 or a maximum of 40 CBFs (40x100KB = 4MB) in a single transaction.

To ensure the user's anonymity, interaction with the blockchain is provided only through REST APIs. To upload the CBF, users need to include their temporary access token with the query. The query mechanism does not require any access authentication. The REST APIs are provided by HAs, which imply that multiple identical APIs are available.

Figure 6 shows the process of uploading relevant information to the blockchain, along with the main actors involved. In general, the interaction involves three sub-processes. Step 1) The HA issues a temporary access token by providing corresponding HA credentials to the blockchain in a transaction via a peer node. The peer node validates the credentials and logs the transaction in the blockchain to mark the issuing of a token. The HA receives the temporary access token and transmits the token to the appropriate user (User A in this case). Step 2) After receiving the token, User A may upload its CBF using the REST API with the appropriate access token. The REST API then forwards the request and responds with the transaction status. Step 3) In an event in which User B wishes to check for potential contact with positive cases, User B can interact with the REST API with a `checkExposure(QBF)` message, which includes its QBF. The REST API forwards the query to the blockchain for checking if the QBF contains potential contacts with positive cases. The REST API messages `wasExposed` to User B to indicate that User B was in contact with an identified positive case.

4.6 Contact verification process

The contact verification process is performed through a smart contract at the blockchain. Each day[5] the app combines all the DBFs into a single QBF. The user also appends the date of the oldest Bloom filter, T_{old}, used in the query. The blockchain takes this query and runs a search through the blocks, trying to match any entry in the QBF with existing CBF transaction entries. Note that the search is restricted to only transactions following the T_{old} date[6]. This search equates to finding the intersection of the two equal-sized filters CBF and QBF constructed with equal number of hashes. This is done by performing a bitwise-AND operation on CBF and QBF to approximate their set intersection. Let t denote the number of bits set in the intersection set, FPR for the intersection set is $= (t/m)^k$. Since t is always less than or at most equal to the set bits in any of CBF and QBF, the FPR for intersection set is $\leq$ FPR for CBF, and is $\leq$ FPR for QBF [30]. The blockchain returns the appropriate response matched or not matched based on the number of set bits in the intersection set.

5 Comparison

We introduced three architectures commonly used for designing digital contact tracing apps in Section 2, and discussed the design of our proposed solution in the previous section. In this section, we compare the salient features of our proposed solution with representative apps from the three architectures.

Table 2 highlights the salient features and their equivalent in selected apps. Our proposed solution, DIMY, generates a temporal ID on the client's device in line with other decentralised and hybrid apps. DIMY is also optimised for storage, both on the client's device as well as the back-end. The design involves storing contact information in fixed-size DBFs. The back-end blockchain only stores a single Bloom filter (CBL, size 100KB) per positive case that has encoded information on the DBFs for the last 21 days. This reduces the storage requirement at the back-end/server considerably when compared with other apps listed in Table 2.

Another salient design question for digital contact tracing apps is where to perform the risk analysis and notification. Apps based on centralised and hybrid architectures perform this step at the centralised server, while apps based on a decentralised architecture perform this locally, on the device. DIMY performs the matching of contact information on the back-end blockchain. However, the blockchain is not able to infer any extra information as the matching is performed on contact information encoded in Bloom filters.

5. For scalability, the query is performed in a 24-hour cycle from the time of app installation.

6. T_{old} date can be a maximum of 21 days old, thus any CBF stored at the blockchain that is older than 21 days is not matched. This automatically takes care of CBFs pertaining to COVID-19 cases that are no longer infectious.

On the other hand, our proposed solution is device-centric in the sense that it performs most of the privacy-preserving operations on the device. This includes EphID generation, computing k-out-of-n shares and broadcasting these shares using BLE messages, and encoding received contact information on DBL after enough shares have been received to construct a shared secret. In comparison, apps such as TraceTogether and CovidSafe (AU) only involve the advertisement of IDs received from the centralised server. Desire also uses the Diffie-Hellman key exchange and the generation of local IDs on the devices. In contrast, Desire uploads the shared secrets collected by a user who has been diagnosed with COVID-19 to a server for matching, to be performed at a later stage. DIMY requires uploading the least amount of data when compared with other apps. A single 100KB sized CBL is uploaded from a COVID positive client to the blockchain. This is in contrast with uploading all contact IDs, required on apps that follow the centralised architecture, and uploading multiple seeds or the PETs table on apps that use decentralised and the hybrid architectures. DIMY also requires client devices to upload QBL, a Bloom filter for matching transactions stored on the blockchain. DIMY client devices only download the results of the risk analysis in the form of a binary notification similar to centralised and hybrid apps. Apps based on the decentralised architecture involves the downloading of either seeds/chirps from the server in order for matching to be performed on the devices.

6 Security and Privacy Analysis

This section is dedicated to an analysis of security and privacy guarantees provided by the DIMY design of protocol.

6.1 Threat Model

In this section, we describe the adversaries considered in the design of the DIMY protocol and the risks that they pose. We categorise the adversaries into three groups, users, back-end developers/administrators and the government. i) *Users* have access to in-app information as well as passive information captured through eavesdropping. App users are also assumed to have access to the open-source app code. Furthermore, we assume users can only have access to data stored on other smartphones through theft or coercion. ii) *Backend administrators/developers* have access to all data received and stored at the backend server. iii) *The government* can access any information stored on individual smartphones or the backend server through subpoenas to investigate a group or individual user of the app.

6.2 Security and Privacy Analysis

Requirements:

1) Completeness: If a person receives an alert message "match", then he/she was definately in proximity to a COVID positive patient.
2) Soundness: If a person was not in the proximity of a COVID positive patient, then there is only a negligible probability that he/she will receive an alert.
3) Type I Privacy: No information is revealed about a patient who has tested positive.

TABLE 2
Comparison of DIMY with other protocols

Salient Features	DIMY	Centralised (BlueTrace)	Decentralised (PACT-East, DP-3T)	Hybrid (Desire)
ID generation	Client devices	Server	Client devices	Client devices
Storage on devices	Encounter encoded in Bloom filters	Received IDs from close contacts	Received IDs (chirps) from close contacts	EphIDs and two PETs tables
Storage on server/back-end	Encounter encoded in Bloom filters for positive cases	Mapping of IDs, Complete list of contact IDs for positive cases	Hourly seeds and time for positive cases	PETs for positive cases
Processing on devices	ID generation, Diffie-Hellman key generation, k-out-of-n secret sharing, Bloom filter encoding	Minimal processing	Hourly seed and chirp generation, Chirp matching and risk analysis	ID generation Diffie-Hellman exchange
Processing on server/back-end	Blockchain matching for at-risk users	Risk analysis and ID matching	Minimal processing	Risk analysis PETs matching
Data upload	Bloom filter for positive cases Query Bloom filter for other users	All contact IDs captured for a positive case	Seeds, timing information for a positive case	PETs table for positive case
Data download	Result (yes/no) from blockchain	Periodic download of new IDs	Seeds,timing information for all positive cases	Result of risk analysis
Risk Analysis & notification	Performed on Blockchain	Performed on server	Performed on devices	Performed on server

4) Type II Privacy: Even if the data present on a device is compromised, it does not leak information about individuals who were in close proximity of the device.

If the key exchange and secret sharing mechanisms are secure, and the Bloom filter and the blockchain implementation are correct, then the protocol achieves completeness, soundness, Type I and Type II privacy.

Completeness:

A user who wants to check whether he/she was in close proximity with a COVID positive individual sends their QBF to the back-end. This is matched against the CBF on the blockchain. Since the check is performed by a smart contract run by peer nodes, it returns a match only if a match exists. This is ensured by the set membership propoerties given by the Bloom filter and the assumption that the blockchain implementation is correct. We note that the QBF is sent through a secure channel.

Soundness:

An attacker who was not in close proximity with a COVID positive patient will receive a match with negligible probability (as computed by Equation 1). This is also ensured by the property of the Bloom filter and the assumption that the blockchain implementation is correct. The blockchain stores only CBFs, Bloom filters that encode the encounter identifiers of COVID positive patients. An attacker, who was not in proximity with a COVID positive patient, cannot construct an encounter identifier for a QBF that will match with the CBF. This is because of the properties of decisional Diffie-Hellman and the hash functions used for encoding encounters in the CBF.

Type I privacy:

The blockchain stores only CBF. A user who uploads a QBF to the blockchain and receives a match knows that he/she has been in contact with at least one COVID positive patient but cannot say which one, due to the design of the Bloom filters. Here we ignore the case in which a person has been in contact with only one person during the last 21 days or uploads only one entry QBF and receives a ”match”. In such a case, the identity of the COVID positive patient is known.

Type II privacy:

When a device is compromised, the DBF is revealed. Since the *EncID* and *EphID*s are not known, and the secrets corresponding to the *EphID*s of the device X_{At} are not known, the attacker cannot know the identity of users in close proximity with the device.

TABLE 3
Possible attacks on digital contact tracing (C=Centralised, D=Decentralised, H=Hybrid)

Attacks	DIMY	C (BlueTrace)	D (PACT-East, DP-3T)	H (Desire)
Replay	×	✓	✓	×
Relay	✓	✓	✓	✓
Device Tracking	✓	✓	✓	✓
Carryover	✓	✓	✓	×
Location Confirmation	×	✓	✓	×
Enumeration	×	✓	×	×
Denial of Service	✓	✓	✓	✓
Linkage	×	✓	✓	×
Social Graph	×	✓	✓	✓

6.3 Privacy Protection

In this section, we discuss the DIMY’s privacy protection properties. In our solution, we adopted blockchain as the back-end service. As sensitive data covering encounter information is first encoded in Bloom filters by clients before uploading to the chain, there is implicit privacy protection against possible back-end breaches (discussed earlier in the threat model). The one-way hash mapping involved in the Bloom filter also significantly reduces the back-end’s ability to construct social graphs based on the diagnosed user’s contacts. The only information the back-end can infer is an estimate of the number of contacts encoded in the CBL uploaded by a positive case, without identifying who these contacts are. On the other hand, users can query the blockchain for matching using QBLs. The result of the query is a simple binary decision that does not reveal which of the encounters in QBL have matched. As users do not have direct access to the CBL stored in the blockchain, they are unable to extract any sensitive information.

The devices, on the other hand, only broadcast shares of the cryptographically generated EphID via BLE advertisements in the contact phase. These pseudo-identifiers cannot be matched back to the real identities of the users unless an adversary accumulates a significant amount of auxiliary information, such as location information obtained by hacking GPS, or eavesdropping on WiFi or other sensors. All encounter information is deleted once it is encoded in the DBF, hence protecting data in case a device gets physically stolen or the user is forced to reveal app data under coercion.

Another potential privacy concern around contact tracing apps is known as the function creep [11]. Function creep refers to the evolution of the app to include functionalities other than the original ones, i.e., the app has the potential to be turned into an instrument of mass surveillance, violating human rights. Thus, it is necessary to analyse the privacy of the proposed app from a function creep perspective.

In the proposed protocol, temporary IDs (i.e., EphIDs) generated by devices are first used to construct encounter IDs that are then encoded in the Bloom filter, which stops underlying linkages from being created between the temporary IDs and concrete IDs in the real world. This binary data encoded in a Bloom filter becomes semantically meaningless to any other user, and even the back-end cannot associate the reported data with an infected person or any specific individual.

Lastly, in the proposed protocol, the blockchain is adopted on the back-end, which provides transparency on the integrity and trustworthiness of the data stored on the chain. Thus, the server is unable to extract any extra information that could assist a compromised back-end to use the stored data for any other purpose. However, the DIMY protocol is also susceptible to privacy attacks launched by malicious users who may use a modified application to collect other contextual information regarding the contacts. Multiple malicious users may work together, combining their information, to collect a large number of recorded broadcasts with metadata on time and location, etc.

6.4 Resilience against attacks

In this section, we will explore the resilience of our proposed design against common attacks launched against digital contact tracing apps.

6.4.1 Replay attacks

In this type of attack, the goal of an adversary is to inject malicious contact information entries such that these entries result in false positives[7] during the contact verification stage. An adversary can capture the advertised messages by a user's device and replays these at another location later on. Our proposed solution provides a safeguard against such attacks by using the secret sharing scheme. An adversary must capture at least k shares of a message, before taking these shares to another location for rebroadcasting. This may result in several recipients using these shares to form contact information in their logs. However, in order to be counted as false positives, the originator of the messages must also have matching entries in their logs. The only way this attack would be possible is if the adversary moves back and forth between two different locations, collecting shares and rebroadcasting these to ensure the existence of symmetric contact information.

The design of our proposed solution renders it impossible, as the adversary has to remain at a particular location for at least k minutes before moving to another location to advertise these collected shares. Assume that the adversary can collect k shares advertised by Alice at location 'A', and rebroadcast it at location 'B'. Once the adversary returns to location 'A' with the shares collected from location 'B', the advertised $EphID$ for user Alice has changed (it has a rotation time of 30 minutes), so it would not result in the storage of symmetric contact information.

6.4.2 Relay attacks

An adversary's objective during a relay attack is the same as it is in a replay attack. An adversary can capture a user's advertised shares and immediately relay the captured message at the same location, extending the range of the message. The adversary thus acts as a relay, transmitting shares that it manages to capture.

Our proposed solution is susceptible to relay attacks that are inherent to all schemes using BLE messages. It is possible to rebroadcast shares such that two users, Alice and Bob, have symmetrical contact information even though they were not in direct contact with each other. We point out that both Alice and Bob have to be in direct communication range of the adversary for k minutes to obtain symmetric contact information. As a consequence, if either Alice or Bob tests positive, the other user would be informed of a 'false positive' close contact.

6.4.3 Device tracking

The adversary's goal in this type of attack is to exploit the fact that most digital tracing apps use BLE and BLE information broadcasts can be used to track a particular device. A passive listening device can listen for BLE advertisements/connections and transfer these to a central tracking server. The server can diffuse information from multiple tracking devices to estimate the position and movement pattern of the device being tracked. It is thus trivial to track a device that is advertising BLE messages while there is an identifier that can be associated with that device. In regular communications, the Bluetooth MAC address is randomised for a short period to limit this tracking. In our proposed solution, we use chunks of $EphIDs$ that are different from each other and use the $EphID$ hash to link all these shares together. An adversary can use the $EphID$ hash in combination with the randomised MAC address to perform limited tracking.

6.4.4 Location confirmation

The adversary's goal is to discover the presence of a user at a known location. This is accomplished by linking contextual information, such as the mobile phone model used in BLE advertisements in apps that are based on centralised contact tracing architectures. This type of attack is not possible in our proposed protocol, due to the use of ephemeral identifiers and the suppression of other information that links the device with a particular user.

6.4.5 Enumeration Attack

This attack aims to estimate the number of users who have uploaded their contact tracing data after testing positive with COVID-19. In our proposed protocol, the encounter data is first encoded in Bloom filters before being stored on the blockchain. A user is allowed to query the

7. A false positive occurs when the digital app indicates a close contact with a positive case, even though that contact has not occurred.

blockchain for matching any encounter record, without revealing the records stored on the blockchain. A malicious user is thus unable to launch an enumeration attack. Note that as the HAs authorise all uploads of CBFs to the blockchain, and there are multiple HAs that exist in the system, they can collude with each other to arrive at the total number of COVID-19 cases that have uploaded CBFs to the blockchain.

6.4.6 *Denial of Service*

In this type of attack, an adversary generates fake advertisements to consume the storage and battery resources of other devices. Digital contact tracing apps are prone to this attack irrespective of their underlying architecture. In our proposed solution, an adversary can force other devices to store fake encounter information by advertising multiple $EphIDs$ instead of using only two identifiers.

6.4.7 *Deanonymisation/Linkage*

An adversary aims to deanonymise a user based on the information it can collect either through the system or by using a side-channel. This attack can be launched to deanonymise close contacts or to identify users who have tested positive. This type of attack is not possible in our proposed solution as information regarding close contacts and positive cases is not directly shared with other users. The query mechanism through the blockchain does not reveal details of an encounter with a positive case; rather, it simply informs the affected user that they are at risk.

6.4.8 *Carryover attack*

An address carryover attack is possible when the changeover time of a randomised Bluetooth MAC address and the temporary identifier are not synchronised. A listener can thus easily link the multiple Bluetooth MAC addresses advertised within the same identifier's life time. Our proposed solution relies on the simultaneous advertising of two identifiers to enable the correct contact information to be captured (discussed in Section 4.3). This mechanism may result in a carryover attack for tracking purposes, whereby an adversary can associate multiple advertised identifiers with the BT MAC addresses used by a device. An adversary can associate the $EphID$ hash that is being advertised along with random MAC and chunks of the raw $EphID$ to track a user's device, as long as that device is within the communication range.

6.4.9 *Social Graph Analysis*

Social graph construction enables the identification of a person's close contacts. This is an imperative part of manual contact tracing in which a health official conducts interview with a positive case to identify their at-risk close contacts. Digital contact tracing mechanism stores the contact information locally on the user's device. This information is utilised to identify close contacts once a user tests positive. In our proposed solution, we have employed two mechanisms that prevent the construction of social graphs. First, we have made use of ephemeral ID generation on the devices as opposed to ID generation by the server. This means that the back-end blockchain cannot link an $EphID$ with a user. Second, we have employed Bloom filters to hide the contact information of positive cases from the distributed blockchain. The blockchain is thus unable to construct a social graph when a user either uploads their contact Bloom or queries the blockchain using the QBF.

Table 3 summarises this section with details of the attacks that could be launched against various architectures, including against our proposed design.

7 PERFORMANCE EVALUATION

In this section, we present a quantitative evaluation of our proposed backend solution based on blockchain implementation, in terms of throughput, latency and resource consumption. We note that for these experiments we generated synthetic data on the device level and supplied this to the blockchain. Our implementation of the DIMY app and its GUI is part of our future work.

7.1 Implementation details

We implemented a proof-of-concept of our proposed framework using Hyperledger Fabric v2.0, as it allows flexibility in modelling Bloom filters in a permissioned blockchain environment. We opted to use a permissioned blockchain to control the app user's access by regulatory organisations, such as the health authority. We consider the standard configuration of a solo orderer node with one communication channel as the consensus mechanism. We ran our experiments on a single machine with 12 cores of CPU and 64GB of memory, running Ubuntu Linux 18.04 LTS. We implemented the core functions of DIMY transactions as chaincodes written in the Go programming language. We selected a native Go implementation of the Bloom Filter v2.0.3[8] and a non-cryptographic murmur hashing function for Go to implement the Bloom filter functionality in the Hyperledger Fabric. We benchmarked our proof-of-concept implementation using Caliper v0.3.2[9], an official tool from the Hyperledger foundation that allows blockchain designers to measure the performance of the implementation of a specific blockchain. We measured the performance per second and repeated the measurements for 30 seconds. We noted the performance of our implementation in terms of throughput, latency, CPU and memory consumption.

7.2 Results

7.2.1 *Throughput and latency for blockchain operations*

In this experiment, we examined the throughput and latency of different DIMY blockchain transactions of uploading CBF, token issue and querying through QBF, when a load of 50 tx/second was sent to the blockchain. We define throughput as the rate at which transactions are successfully executed and latency as the time required to complete the transactions. Please note that although a complete processing cycle of our architecture includes the serial execution of multiple DIMY transactions, we examined the throughput and latency only for each individual transaction. For instance, we assume that the CBF is already uploaded to the blockchain and we only measure the throughput and latency for executing query QBF. This definition does not consider the network latency, which could be impacted by different external factors.

8. https://github.com/willf/bloom
9. https://hyperledger.github.io/caliper/

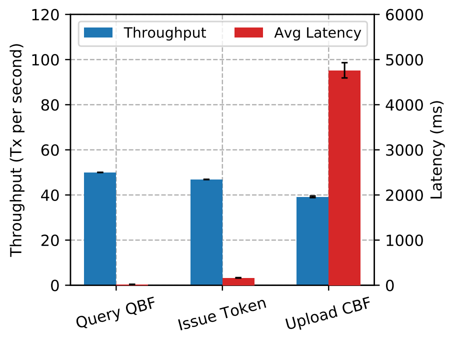


Fig. 7. Comparison of the average latency and the throughput for blockchain operations in caliper, using a load of 50 transactions per second.

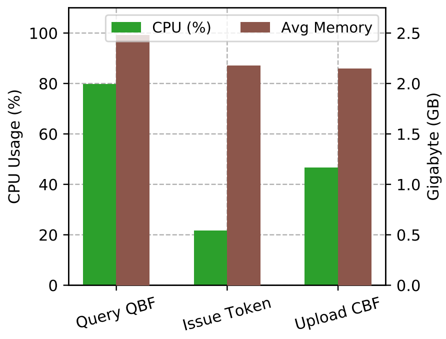


Fig. 8. Comparison of CPU and memory consumption for the execution of querying QBF, issuing an access token and uploading CBF.

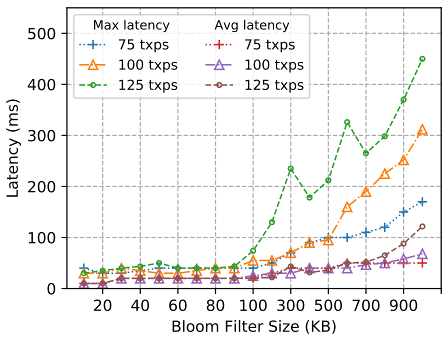


Fig. 9. The maximum and average latency of querying QBF with differently sized Bloom filters.

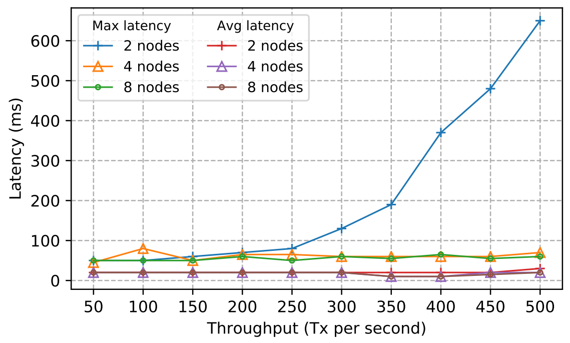


Fig. 10. Throughput and latency of querying QBF with the size of 100KB in different transaction send rates and number of Hyperledger peer nodes.

We plot the results in Figure 7. Among three DIMY blockchain transactions, QBF upload and matching is the transaction with the lowest latency, while uploading CBF has the highest latency, at around 4700 ms. Note that although we set a constant transaction send rate of 50 tx/second, issue tokens and upload CBF transactions failed to deliver the same throughput rate. The latency for uploading 50 simultaneous CBFs is the highest when compared with other operations, as this involves transaction insertions and consensus operations at the blockchain. QBF matching, on the other hand, performs very well in terms of latency and throughput. We note that uploading CBF is only performed once for each identified COVID-19 positive case, while QBF upload and matching is performed once in a 24-hour cycle by each user.

7.2.2 *CPU and memory consumption for blockchain operations*

We compare the CPU and memory consumption of different operations and show the results in Figure 8. A Caliper monitor was utilised to capture CPU and memory usage when we applied a load of 50 tx/second to the Hyperledger Fabric network for 30 seconds. The results show that although there seems to be no significant difference in average memory consumption, QBF matching consumes the highest CPU percentage (about 80%), when compared with other operations. This result highlights that to host the backend blockchain, the back-end's memory and CPU requirements are the key design factors, as QBF matching is the most used operation on the backend.

7.2.3 *Latency of querying QBF with different sized Bloom filters*

The following experiment aims to examine the effect of using different sized Bloom filters on the resulting latency of QBF matching operations on the blockchain. We executed different transaction rate loads, namely 75, 100 and 125 tx/second, noting both maximum and the average latency for QBF sizes varied from 10KB to 100KB. The results shown in Figure 9 demonstrate that the average latency remains less than 100 ms for all QBF sizes less than 900KB, while the maximum observed latency remains less than 100 ms for QBF sizes up to 100KB. The maximum latency starts increasing considerably, especially for higher transaction rates, if the size of the QBF is increased beyond 100KB. This result shows that our chosen value of 100KB for Bloom filters is optimal to minimise the maximum observed latency for different transaction rates.

7.2.4 *Throughput and latency analysis*

Lastly, we investigate the throughput and latencies for querying QBF with different number of Hyperledger peer nodes (2 , 4 and 8 nodes) and plot the results in Figure 10. During this experiment, we started from 50 QBF tx/second and gradually increased the send rate up to 500 tx/second. In this experiment, we noticed that the blockchain was still able to deliver increasing throughput in line with the transaction send rate. Besides, maximum latency remains below 100 ms for transaction rate of up to 250 tx/second for all explored cases, after which there is an observed increase for 2 nodes Hyperledger. However, the average latency stays low (less than 50 ms) for all cases. We note that this observed performance is achieved with the hardware resources used for this proof of concept described in Section 7.1, which can be easily scaled up in the actual deployment of the proposed architecture.

8 Conclusion

In this paper, we have presented the design and security and privacy evaluation for DIMY, a privacy-preserving digital contact tracing protocol. Our protocol design integrates several privacy preserving techniques, assuming both malicious users and the back-end as the threat model. We employed a Bloom filter to enhance privacy protection as well as to considerably cut down storage requirements both on the client's device and the back-end.

Our protocol is resilient against most of the security and privacy attacks commonly launched against digital contact tracing apps. The proposed protocol incurs negligible overheads and supports low latency operations on the backend side, as demonstrated in our performance evaluations. The development of the open-source app and its GUI is part of our future work.

ACKNOWLEDGMENTS

This work has been supported by the Cyber Security Cooperative Research Centre Limited (CSCRC), whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.

REFERENCES

[1] J. Bay, J. Kek, A. Tan, C. S. Hau, L. Yongquan, J. Tan, and T. A. Quy, "Bluetrace: A privacy-preserving protocol forcommunity-driven contact tracing across borders," https://bluetrace.io/static/bluetrace_whitepaper-938063656596c104632def383eb33b3c.pdf, 2020.

[2] R. L. Rivest, J. Callas, R. Canetti, and et al., "The pact protocol specifications," *Technical report*, vol. 0.1, April, 2020. [Online]. Available: https://pact.mit.edu/wp-content/uploads/2020/04/The-PACT-protocol-specification-ver-0.1.pdf

[3] Apple, "Privacy preserving contact tracing," https://www.apple.com/covid19/contacttracing, 2020.

[4] Google, "Exposure notification api," https://www.google.com/covid19/exposurenotifications/, 2020.

[5] P. H. O'Neill, T. Ryan-Mosley, and B. Johnson, "A flood of coronavirus apps are tracking us. now it's time to keep track of them," https://www.technologyreview.com/2020/05/07/1000961/laun-ching-mittr-covid-tracing-tracker/, 2020.

[6] N. Ahmed, R. A. Michelin, W. Xue, S. Ruj, R. Malaney, S. S. Kanhere, A. Seneviratne, W. Hu, H. Janicke, and S. K. Jha, "A Survey of COVID-19 Contact Tracing Apps," *IEEE Access*, vol. 8, pp. 134 577–134 601, 2020.

[7] S. Vaudenay, "Centralized or decentralized? the contact tracing dilemma," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 531, 2020.

[8] ——, "Analysis of dp3t: Between scylla and charybdis," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 399, 2020.

[9] OpenTrace, "Opentrace," https://github.com/opentrace-community.

[10] COVIDSafe, "Covidsafe," https://github.com/AU-COVIDSafe.

[11] C. Troncoso and et.al., "DP-3T," https://github.com/DP-3T.

[12] C. Castelluccia, N. Bielova, A. Boutet, and et al, "Desire: A third way for a european exposure notification system leveraging the best of centralized and decentralized systems," *Technical report*, vol. hal-02570382, 2020.

[13] W. Beskorovajnov, F. Dörre, G. Hartung, and et. al, "Contra corona: Contact tracing against the coronavirus by bridging the centralized–decentralized divide for stronger privacy," Cryptology ePrint Archive, Report 2020/505, 2020.

[14] ROBERT, https://github.com/ROBERT-proximity-tracing/document, 2020.

[15] TCN Coalition, "TCN protocol for decentralized, privacy-preserving contact tracing," https://github.com/TCNCoalition/TCN.

[16] The DP3T Consortium, "DESIRE: A practical assessment," https://github.com/DP-3T/documents/blob/master/Security%20analysis/DESIRE%20-%20A%20Practical%20Assessment.pdf.

[17] H. Xu, L. Zhang, O. Onireti, Y. Fang, W. B. Buchanan, and M. A. Imran, "Beeptrace: Blockchain-enabled privacy-preserving contact tracing for covid-19 pandemic and beyond," 2020.

[18] W. Lv, S. Wu, C. Jiang, Y. Cui, X. Qiu, and Y. Zhang, "Decentralized blockchain for privacy-preserving large-scale contact tracing," *CoRR*, vol. abs/2007.00894, 2020. [Online]. Available: https://arxiv.org/abs/2007.00894

[19] G. Avitabile, V. Botta, V. Iovino, and I. Visconti, "Towards defeating mass surveillance and sars-cov-2: The pronto-c2 fully decentralized automatic contact tracing system," Cryptology ePrint Archive, Report 2020/493, 2020.

[20] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information theory*, vol. IT-22, pp. 644–654, 1976.

[21] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[22] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[23] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[24] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Tech. Rep., 2019.

[25] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[26] V. Dedeoglu, R. Jurdak, A. Dorri, R. C. Lunardi, R. A. Michelin, A. F. Zorzo, and S. S. Kanhere, *Blockchain Technologies for IoT*. Singapore: Springer Singapore, 2020, pp. 55–89.

[27] Ethereum, "Ethereum.org," https://ethereum.org/, 2020.

[28] Hyperledger, "Hyperledger – open source blockchain technologies," https://www.hyperledger.org/, 2020.

[29] D. Boneh, "The decision diffie-hellman problem," in *Algorithmic Number Theory*, J. P. Buhler, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 48–63.

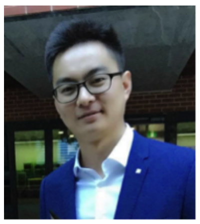
[30] O. Papapetrou, W. Siberski, and W. Nejdl, "Cardinality estimation and dynamic length adaptation for bloom filters," *Distributed Parallel Databases*, vol. 28, pp. 119–156, 2010.

[31] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[32] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20, 000 transactions per second," *CoRR*, vol. abs/1901.00910, 2019.

Nadeem Ahmed received M.S. and Ph.D. degrees in computer science from the UNSW, Sydney, Australia, in 2000 and 2007, respectively. He is currently working as a senior research fellow at the Cyber Security Cooperative Research Centre (CSCRC), Australia. Earlier, he worked as head of the Computing Department at the School of Electrical Engineering and Computer Science, NUST, Pakistan. His research interests include cyber security, IoT, wireless sensor networks, and software-defined networking.

Regio A. Michelin received M.S. and Ph.D. degrees in computer science from the Pontifical Catholic University of Rio Grande do Sul, Brazil, in 2014 and 2019, respectively. He is currently working as research fellow at the Cyber Security Cooperative Research Centre (CSCRC), Australia. His research interests include blockchain, cybersecurity, and IoT.



Wanli Xue received Ph.D. degree from the School of Computer Science and Engineering, UNSW, Australia. He is currently a Research Fellow at the Cyber Security Cooperative Research Centre (CSCRC) and UNSW, Australia. His research interests include security and privacy issues in cyber physical systems and IoT, including highly efficient privacy-preserving techniques for IoT as well as IoT-related sensing systems and data analytic services.

Guntur Dharma Putra received his bachelor degree in Electrical Engineering from Universitas Gadjah Mada, Indonesia, in 2014. He received his master's degree in Computing Science from the University of Groningen, the Netherlands, in 2017. He is currently a Ph.D. candidate at UNSW, Sydney, Australia. His research interests cover distributed systems and the IoT. He also looks into blockchain applications for securing IoT. Guntur is a student member of the IEEE.

Sushmita Ruj received her Masters and PhD in Computer Science from the Indian Statistical Institute. She is currently a Senior Research Scientist at CSIRO Data61, Australia. She is also an Associate Professor at Indian Statistical Institute, Kolkata. Her research interests include blockchains, applied cryptography, and data privacy. She serves as a reviewer of Mathematical Reviews, and an Associate Editor of Elsevier Journal, Information Security and Applications. She is a recipient of the Samsung GRO award, NetApp Faculty Fellowship, Cisco Academic Grant and IBM OCSP grant. She is a Senior Member of the ACM and IEEE.

Salil S. Kanhere received his M.S. and Ph.D. degrees from Drexel University in Philadelphia. He is a Professor of Computer Science and Engineering at UNSW Sydney, Australia. His research interests include the IoT, blockchain, pervasive computing, cybersecurity and applied machine learning. He is a Senior Member of the IEEE and ACM, an Humboldt Research Fellow and an ACM Distinguished Speaker. He serves as the Editor in Chief of the Ad Hoc Networks journal and as Associate Editor of IEEE TNSM, COMCOM and PMC. He has served on the organising committee of several IEEE/ACM international conferences.

Sanjay K. Jha is a Full Professor and Director of the Cybersecurity and Privacy Lab at the School of Computer Science and Engineering at the UNSW, Australia. He leads UNSW in the Cybersecurity Cooperative Research Centre. His research activities are primarily focused on Wireless Mesh/Sensor Networks (IoT), and Network Security. He is the principal author of the book ”Engineering Internet QoS” and a co-editor of the book ”Wireless Sensor Networks: A Systems Perspective.” His editorial affiliations include the IEEE TMC and TDSC.