# Glass-Vault: A Generic Transparent Privacy-preserving Exposure Notification Analytics Platform

LORENZO MARTINICO, University of Edinburgh, Scotland
AYDIN ABADI, University College London, England
THOMAS ZACHARIAS, University of Edinburgh, Scotland
THOMAS WIN, University of the West of England, Bristol, England

The highly transmissible COVID-19 disease is a serious threat to people's health and life. To automate tracing those who have been in close physical contact with newly infected people and/or to analyse tracing-related data to better understand the spread of the disease, researchers have proposed various ad-hoc solutions that can be installed on users' smartphones. Nevertheless, the existing solutions have two primary limitations: namely (1) lack of generality: for each type of analytic task, a certain kind of data needs to be sent to the analyst; (2) lack of transparency: the provider of data is not necessarily the infected individuals, whose data is shared without fine-grained and direct consent. In this work, we present "Glass-Vault", an extension of existing Exposure Notification protocols that addresses both limitations simultaneously. Glass-Vault lets an analyst run authorised programs over the collected data of infectious users, without learning the input data. To construct Glass-Vault, we build a new variant of "Steel" (Bhatotia *et al.*, PKC'21), a Functional Encryption scheme which supports Stateful and Randomised functions and relies on Trusted Execution Environments. This new variant offers two additional properties: it is *dynamic* and *decentralised*. We prove the security of this "Dynamic and Decentralised Steel" and Glass-Vault in the Universal Composability model. Glass-Vault is the first platform that offers all the above features. As a sample application, we illustrate how it can be used to generate infection heatmaps.

**ACM Reference Format:**
Lorenzo Martinico, Aydin Abadi, Thomas Zacharias, and Thomas Win. 2022. Glass-Vault: A Generic Transparent Privacy-preserving Exposure Notification Analytics Platform. In . ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The Coronavirus (COVID-19) pandemic has been significantly affecting individuals' personal and professional lives as well as the global economy. The risk of COVID-19 transmission is immensely high among people in close proximity. Tracing those individuals who have been near recently infected people (a.k.a. contact tracing) is one of the vital approaches to efficiently diminishing the spread of COVID-19 [22]. Contact tracing allows to identify and instruct only those who have potentially contracted the virus to self-isolate, without having to require an entire community to do so. This is crucial when combating a pandemic, as widespread isolation can have destructive effects on people's (mental and physical) well-being and countries' economies. To digitise the tracing process, researchers have proposed numerous solutions that can be installed on users' smartphones, most notably the family of Automated "Exposure Notification" systems. To engender sufficient adoption throughout the population [**?** ], most of the proposed systems attempt to implement privacy-preserving techniques, such as hiding infected users' contact graphs, or adopting designs that prevent tracking of non-infected users [30]. There have also been a few ad-hoc privacy-preserving solutions that help analyse other user-originated data, e.g., location history to map the virus clusters [15], or scanning QR codes to notify those who were co-located with infected individuals [28].

This kind of data analytics can play a crucial role in better understanding the spread of the virus and ultimately helps inform governments' decision-making (e.g., to close borders, workplaces, or schools) and enhance public health advice; especially, when the analytics result is combined with the general public's health record, e.g., a region's average age, or people's previous exposure to infections. Despite the importance of this type of solution, only a few have been proposed that can preserve users' privacy. However, they suffer from two main limitations.

Firstly, they *lack generality*; in the sense that for each type of data analysis, a certain data type has to be sent to the analyst which ultimately (i) limits the applications of such solutions, (ii) increases user-side computation, communication, and storage costs, and (iii) demands a fresh cryptographic protocol to be designed, defined, and proven secure for every operation type. It is desirable that all kinds of user data, independently of their format, could be securely collected and transmitted through a unified protocol. In concrete terms, such a unified protocol would provide a way for scientists and health authorities to focus on the analysis of data without having to design secure protocols. Moreover, users would not require to install multiple applications on their devices to concurrently run data capturing programs (which would potentially cause issues especially for users with resource-constrained devices).

Secondly, existing solutions *lack transparency*; meaning that users are not in control of what kind of sensitive data is being collected about them (e.g. the Israeli contact tracing system [7]). Also, in some of these schemes, the data are not originated directly from the users but from a third-party data collector (e.g., a mobile service provider) which aggregates the data and provides them to an analytic service without having fine-grained and direct consent of the users. This is not desirable and may even be against privacy legislation since users, as data owners, must be able to authorise the type of computations executed on their data.

**Our Contributions.** To address the aforementioned limitations, in this work, we propose a platform called "Glass-Vault". Glass-Vault is an extension of regular privacy-preserving decentralised contact tracing, which additionally allows infected users to share sensitive (non-contact tracing) data for analysis. Glass-Vault is a generic platform, as it is (data) type agnostic and supports *any secure computation* that users authorise. It also offers transparency and privacy, by allowing users to consentually choose what data to share, and forcing an analyst to execute only those computations authorised by a sufficient number of users, without getting access to the raw users' data. In this protocol, the analyst is considered an active adversary who may want to learn additional information about the data beyond what it is authorised to compute, and can collude with other users of the contact tracing protocol. We formally define Glass-Vault in the Universal Composability (UC) paradigm and prove its security. Specifically, we first define an ideal functionality that captures an "Analysis-augmented" extension of standard exposure notification ($\text{EN}^+$) and then show that Glass-Vault UC-realises said functionality. To attain its goals, Glass-Vault relies on extending the Steel protocol of Bhatotia et al. [12] for "Functional Encryption for Stateful and Randomised Functionalities" (FESR) into *DD*-Steel, a protocol that realises the "Dynamic Decentralised FESR" (*DD*-FESR) functionality. Our extension allows the functional key to be generated in a distributed fashion while letting any authorised party freely join the key generator committee. *DD*-FESR and *DD*-Steel can be of independent interest. As a concrete application of Glass-Vault, we show how it can be utilised to help the analyst identify the virus clusters, a.k.a., virus heatmaps.

## 2 BACKGROUND

### 2.1 Universal Composibility

Universal Composibility (UC), introduced by Canetti [16], is a security paradigm enabling the security analysis of cryptographic protocols. Informally, in this paradigm, the security of a protocol is shown to hold by comparing its execution in the real world with an ideal world execution, given a trusted *ideal functionality* that precisely captures the appropriate security requirements. A bounded environment $\mathcal{Z}$, which provides the parties with inputs and schedules the execution, attempts to distinguish between the two worlds. To show the security of the protocol, there must exist an ideal world adversary, often called a simulator, which generates the protocol's transcript indistinguishable from the real protocol. We say *the protocol UC-realises the functionality*, if for every possible bounded adversary in the real world there exists a simulator such that $\mathcal{Z}$ cannot distinguish the transcripts of the parties' outputs. Once a protocol is defined and proven secure in the UC model, other protocols, that use it as a subroutine, can securely replace the protocol by calling its functionality instead, thanks to the UC composition theorem.

Later, Canetti et al. [18] extend the original UC framework and provide a generalised UC (GUC) to re-establish UC's original intuitive guarantee even for protocols that use *globally* available setups, such as public-key infrastructure (PKI) or a common reference string (CRS), where all parties and protocols are assumed to have access to some global information which is trusted to have certain properties. GUC formalisation aims at preventing bad interactions even with adaptively chosen protocols that use the same setup. Badertscher et al. [10] proposed a new UC-based framework, UC with Global Subroutines (UCGS), that allows the presence of global functionalities in standard UC protocols.

### 2.2 Privacy-Preserving Contact Tracing

**Centralized vs Decentralized Contact tracing.** Within the first few months of the COVID-19 pandemic, a large number of theoretical (e.g., [9, 20, 34, 37]) and practical (e.g., [1, 4–6, 26]) automated contact tracing solutions were quickly developed by governments, industries, and academic communities. Most solutions concentrated around two architectures, so-called "centralised" and "decentralised". To put it simply, the major difference between the two architectures rests on key generation and exposure notification. In a centralised system, the keys are generated from a trusted health authority and distributed to contact tracing users. In the decentralised systems, the keys are generated locally by each user. In both types, information is exchanged in a peer to peer fashion, commonly through Bluetooth Low Energy (BLE) messages broadcast by each user's phone. Once someone is notified of infection, they upload some data to the health authority server. While in centralised systems the uploaded data usually corresponds to the BLE broadcast the infected users' devices listened to, in a decentralised system it will generally be the messages they sent. Since the (centralised) authority knows the identity of each party in the system, it can notify them about exposure. Instead, the decentralised users download the list of broadcasts from exposed users and compare it with their own local lists. Decentralised systems are typically praised for their additional privacy compared to centralised systems (although several attacks are still possible, see [30]). On the other hand, these privacy-preserving properties impair the health authority from running large scale analysis on population infection, whereas such analysis is possible in centralised systems. Despite this, the adoption of decentralised systems such as DP-3T [38] has been more widespread due to technical restrictions and political decisions forced by smartphone manufacturers [8]. There has been much debate on how any effort to the adoption of a more private and featureful contact tracing scheme could be limited by these gatekeepers [9, 39, 40].

**Formalising Exposure Notification in the UC framework.** Canetti et al. [19] introduce a comprehensive approach to formalise the Exposure Notification primitive via the UC framework, showing how a protocol similar to DP-3T realises their ideal functionality. Their UC formulation is designed to capture a wide range of Exposure Notification settings. The modelling relies on a variety of functionalities that abstract phenomena such as physical reality events and Bluetooth communications.

**Beyond Contact Tracing: automating data analysis on population behaviour.** A few attempts have been made to develop additional automated systems which can analyse population trends to better understand the spread of the virus (orthogonal to Contact Tracing), or Exposure Notification systems that do not rely on proximity tracing.

Bruni et al. [15] propose a system to detect the development of virus hotspots based on a scheme which uses homomorphic encryption to compile mobility data collected from mobile phone providers and infection records from health authorities. Lueks et al. [28] design a privacy-preserving “Presence-Tracing” system, which notifies people who were in the same venue as an infected individual. Abramson et al. [2] propose a way to share personal health information beyond the normal contact tracing data in a transparent and privacy-preserving manner.

Biasse et al. [13] design a privacy-preserving scheme to anonymously collect information about a social graph of users which lets the result recipient understand the progression of the virus among the population of users. Günther et al. [27] propose a privacy-preserving scheme between multiple non-colluding servers to help epidemiologists simulate and predict future developments of the virus.

## 2.3 Trusted Execution Environments

Trusted Execution Environments (TEEs) are secure processing environments (a.k.a. *secure enclaves*) that consist of processing, memory, and storage hardware units. In these environments, the residing code and data are isolated from other layers in the software stack including the operating system. TEEs ensure that the integrity and confidentiality of data residing in them are preserved. They can also offer remote attestation capabilities, which enable a party to remotely verify that an enclave is running on a genuine TEE hardware platform. Under the assumption that the physical CPU is not breached, enclaves are protected from an attacker with physical access to the machine, including the memory and the system bus. The first formal definition of TEEs notion was proposed by Pass et al. [33] in the GUC model. This definition provides a basic abstraction called the *global attestation functionality* ($\mathcal{G}_{\text{att}}$, described in Appendix B.1), that casts the core services a wide class of real-world attested execution processors offer. $\mathcal{G}_{\text{att}}$ has been used by various protocols both in the GUC model (e.g., in [36, 41]) and recently in the UCGS model (in [12]).

Some protocols have been proposed to implement Contact Tracing solutions using TEEs [21, 29, 35].

## 2.4 Functional Encryption

Informally, “Functional Encryption” (FE) is an encryption scheme that ensures that a party who possesses a decryption key learns nothing beyond a specific function of the encrypted data. Many types of encryption (e.g., identity-based or attribute-based encryptions) can be considered as a special case of FE. The notion of FE was first defined formally by Boneh et al. [14]. FE involves three sets of parties: $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$. Parties in $\mathbf{A}$ are encryptors, parties in $\mathbf{B}$ are decryptors, and parties in $\mathbf{C}$ are key generation authorities. The syntax of FE is recapped in Appendix A.

An FE scheme can also be thought of as a way to implement access control to an ideal repository [32]. Since the introduction of FE, various variants of FE schemes have been proposed, such as these that (i) support distributed ciphertexts letting joint functions be run on multiple inputs of different parties, (ii) support distributed secret keys that do not require a single entity to hold a master key, or (iii) support both properties simultaneously. In particular (ii) and

(iii) affect the membership of sets **A** and **C**. We refer readers to [3, 31] for surveys of FE schemes. Recently, Bhatotia et al. [12] proposed FESR, a generalisation of FE that allows the decryption of the class of *Stateful* and *Randomised* functions. This class is formally defined as $\mathsf{F} = \{F \mid F : \mathcal{X} \times \mathcal{S} \times \mathcal{R} \rightarrow \mathcal{Y} \times \mathcal{S}\}$, where $\mathcal{S}$ is the set of possible function states and $\mathcal{R}$ is the universe of random coins. They also provide a protocol, called Steel, that realises FESR in the UCGS model and relies on TEEs (as abstracted by $G_{\text{att}}$). The use of FESR is more appealing than other proposed extensions to FE, because it allows a broader class of functions, has a secure realisation in the UC paradigm, and can be extended to support multiple inputs (as we show in Subsection 3.3).

## 3 DYNAMIC AND DECENTRALISED FESR (*DD*-FESR) AND STEEL (*DD*-Steel)

In this section, we present the ideal functionality *DD*-FESR and protocol *DD*-Steel. As we stated earlier, they are built upon the original functionality FESR and protocol Steel, respectively. The formal descriptions presented later in this section highlight in yellow the differences from the originals in [12]. For conciseness, we omit any reference to UC-specific machinery such as session ids unless when required.

### 3.1 The ideal functionality *DD*-FESR

In this subsection, we extend FESR to new functionality called *DD*-FESR which captures two additional vital properties from the functional encryption literature:

- Decentralisation (introduced in [23]): it allows a set of encryptors to be in control of the key generation (rather than a single trusted authority of type **C**). The KeyGen subroutine of the FESR scheme is replaced by a new subroutine KeyShareGen, which can be run by any party A ∈ **A** to produce a "key share". In this case, a decryptor B needs to collect at least $k$ shares for some function F before B is allowed to decrypt. This threshold parameter, $k$, is specified by A when it wants to encrypt, and is unique for each ciphertext, but does not restrict key share generation to a specific subset of **A**.
- Dynamic membership (introduced in [24]): it allows any party to freely join set **A** during the execution of the protocol. In our instantiation, a new party A joins through a local procedure which only requires the public parameters. *DD*-FESR can be instantiated with some prexisting **A** members, or with an empty set that is gradually filled through Setup calls. Our current scheme is premissionless, meaning anyone can register a new party.

**Functionality** *DD*-FESR[F, **A**, **B**, C]

The functionality is parameterised by the randomized function class $\mathsf{F} = \{F \mid F : \mathcal{X} \times \mathcal{S} \times \mathcal{R} \rightarrow \mathcal{Y} \times \mathcal{S}\}$, over state space $\mathcal{S}$ and randomness space $\mathcal{R}$, and by three distinct types of party entities A ∈ **A**, B ∈ **B**, C interacting with the functionality via dummy parties (that identify a particular role).

| State variables | Description |
|---|---|
| $\hat{\mathbf{A}} \leftarrow [\,]$ | List of corrupted As |
| $\hat{\mathbf{B}} \leftarrow [\,]$ | List of corrupted Bs |
| $F_0$ | Leakage function returning the length of the message |
| $\mathsf{F}^+$ | Union of the allowable functions and leakage function, i.e., $\mathsf{F} \cup F_0$ |
| $\mathsf{setup}[\cdot] \leftarrow \mathsf{false}$ | Table recording which parties were initialized. |
| $\mathcal{M}[\cdot] \leftarrow \perp$ | Table storing the plaintext for each message handler |
| $\mathcal{P}[\cdot] \leftarrow \emptyset$ | Table of authorized functions' states for all decryption parties |
| $\mathcal{KS}[\cdot] \leftarrow [\,]$ | Table of key share generator for each (decryptor,function) pair |

```
On message SETUP from $P$:
  if $P$.code = A then $\mathbf{A} \leftarrow \mathbf{A} \parallel P$        ▷ Party membership is determined by the code in the UC identity tape
  else if $P$.code = B then $\mathbf{B} \leftarrow \mathbf{B} \parallel P$
  else return
  setup[$P$] ← true
  send (SETUP, $P$) to $\mathcal{A}$
On message (KEYSHAREGEN, F, B) from A $\in \mathbf{A}$:
  if A $\in \hat{\mathbf{A}}$ then        ▷ The adversary can only block key generation for corrupted parties
    send (KEYSHAREGEN, F, A, B) to $\mathcal{A}$ and await ACK; then
  if (F $\in$ $\mathrm{F}^+$ $\wedge$ setup[A] $\wedge$ setup[B]) then
    $\mathcal{KS}$[B, F] ← $\mathcal{KS}$[B, F] $\parallel$ A        ▷ We store the identity of all parties in A who authorised F for B
    send (KEYSHAREGEN, F, A, B) to B
    if B $\in \hat{\mathbf{B}}$ then send (KEYSHAREGEN, F, A, B) to $\mathcal{A}$
    else send (KEYSHAREGEN, ⊥, A, B) to $\mathcal{A}$
On message (ENCRYPT, x, $k$) from party $P \in \{\mathbf{A} \cup \mathbf{B}\}$:
  if (setup[$P$] $\wedge$ x $\in \mathcal{X}$ $\wedge$ $k$ is an integer) then
    compute h ← getHandle        ▷ Generate a unique index, h, by running the subroutine getHandle
    $\mathcal{M}$[h] ← (x, $k$)
    send (ENCRYPTED, h) to $P$
On message (DECRYPT, h, F) from party B $\in \mathbf{B}$:
  (x, $k$) ← $\mathcal{M}$[h]; y ← ⊥; s ← $\mathcal{P}$[B, F]
  if $|\hat{\mathbf{A}}| \geq k$ then        ▷ If at least $k$ parties are corrupted the adversary can authorise any function
    send (DECRYPT, h, F, x) to $\mathcal{A}$ and receive (DECRYPTED, y, F′, s′)
    $\mathcal{P}$[B, F′] ← s′
  else if $((|\mathcal{KS}[B, F]| \geq k) \wedge (\forall A \in \mathcal{KS}[B, F] : \mathrm{setup}[A]) \wedge \forall A \text{ are distinct})$ then
    // There are at least $k$ functional key shares, all generated by correctly setup parties
    $r \xleftarrow{\$} \mathcal{R}$
    (y, s′) ← F(x, s; r)
    $\mathcal{P}$[B, F] ← s′
  return (DECRYPTED, y)
On message (CORRUPT, $P$) from $\mathcal{A}$:
  if $P \in \mathbf{A}$ then
    $\hat{\mathbf{A}} \leftarrow \hat{\mathbf{A}} \parallel P$        ▷ The functionality needs to keep track of corrupted parties in A to ensure correctness
  if $P \in \mathbf{B}$ then
    $\hat{\mathbf{B}} \leftarrow \hat{\mathbf{B}} \parallel P$
```

## 3.2 The protocol *DD*-Steel

Now, we propose *DD*-Steel, a new protocol that extends the original Steel to realise *DD*-FESR. We first briefly provide an overview of Steel and our new extension, before outlining the formal protocol.

Steel uses a public key encryption scheme, where the master public key is distributed to parties in **A**, and the master secret key is securely stored in an enclave running program $\mathrm{prog}_{\mathrm{KME}}$ on trusted party C. The secret key is then provisioned to enclaves running on parties in B, only if they can prove through remote attestation that they are running a copy of $\mathsf{prog}_{\mathsf{DE}}$. The functional key corresponds to signatures over the representation of a function F and is generated by C's $\mathsf{prog}_{\mathsf{KME}}$ enclave. If party B possesses any such key, its copy of $\mathsf{prog}_{\mathsf{DE}}$ will distribute the master secret key to a $\mathsf{prog}_{\mathsf{FE}[\mathsf{F}]}$ enclave, which can then decrypt any A's encrypted inputs x and will ensure that only value y of $(y, s') \leftarrow F(x, s; r)$ is returned to B, with the function states s and s′ protected by the enclave.

The protocol $DD$-Steel is similar to the original version; however, there are a few crucial differences. C, who is now untrusted, still runs the public key encryption parameter generation within an enclave, and distributes it to a newly joined A or B. Now each party A also generates a digital signature key pair locally, and include a key policy $k$ along with their ciphertext. Note, our current version encrypts $k$ along with the message for convenience, but it would be possible to use a public key policy as a threshold version of Multi-Client Functional Encryption. Party A who wants to authorise party B to compute a certain function will run KeyShareGen(F, B) to generate a key share, which requires signing the representation of F with their local key, and send the signature to B. For B's $\mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}$ enclave to authorise the functional decryption of F, it first verifies that all key shares provided by B for F are valid and each was provided by a unique party in **A**; if all checks are passed, then it will distribute the master secret key to $\mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\mathsf{F}]}$, along with the length of recorded key shares $k_{\mathsf{F}}$. $\mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\mathsf{F}]}$ will only proceed with decryption if the number of key shares meets the encryptor's key policy. Provisioning of the secret key between C and B's $\mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}$ enclave remains as in FESR.

The protocol $DD$-Steel makes use of the global attestation functionality $\mathcal{G}_{\mathsf{att}}$, the certification functionality $\mathcal{F}_{\mathsf{CERT}}$, the common reference string functionality $\mathcal{CRS}$, the secure channel functionality $\mathcal{SC}_R^S$, and the repository functionality $\mathcal{REP}$ that are presented in Appendix B.1, B.2, B.3, B.4, and B.5 respectively. The code of the enclave programs $\mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}$, $\mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}$, $\mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\cdot]}$ in $DD$-Steel is hardcoded with the value of the verification key VK returned by $\mathcal{F}_{\mathsf{CERT}}$, and can be generated during the protocol runtime.

**Protocol** $DD$-Steel$[\mathsf{F}, \mathsf{PKE}, \Sigma, \mathsf{N}, \lambda]$

The protocol is paremeterised by the class of functions F as defined in $DD$-FESR, the public-key encryption scheme PKE denoted as the triple of algorithms (PKE.PGen, PKE.Enc, PKE.Dec), the digital signature scheme $\Sigma$ denoted as the triple of algorithms ($\Sigma$.Gen, $\Sigma$.Sign, $\Sigma$.Vrfy), the non-interactive zero-knowledge protocol N that consists of prover $\mathcal{P}$ and verifier $\mathcal{V}$, and the security parameter $\lambda$.

| State variables | Description |
|---|---|
| $\mathcal{KS}[\cdot] \leftarrow \emptyset$ | Table of function key shares at B |
| $\mathcal{K}[\cdot] \leftarrow \emptyset$ | Table of functional enclave details at B |

**Key Generation Authority** C**:**

```
On message (Setup, $P$):
  if $\mathsf{mpk} = \perp$ then
      send Get to $\mathcal{CRS}$ and receive $(\textsc{Crs}, \mathsf{crs})$
      send GetK to $\mathcal{F}_{\mathsf{CERT}}$ and receive VK
      $\mathsf{eid}_{\mathsf{KME}} \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{install}(\mathsf{C.sid}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}})$
      $(\mathsf{mpk}, \sigma_{\mathsf{KME}}) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{KME}}, (\mathsf{init}, \mathsf{crs}, \mathsf{C.sid}))$
  if $P.\mathsf{code} = \mathsf{A}$ then
      send $(\textsc{Setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ to $\mathcal{SC}_{\mathsf{A}}$
  else if $P.\mathsf{code} = \mathsf{B}$ then
      send $(\textsc{Setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ to $\mathcal{SC}_{\mathsf{B}}$ and receive $(\textsc{provision}, \sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD})$
      $(\mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{sk}}) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{KME}}, (\mathsf{provision}, (\sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}})))$
```

```
        send $(\textsc{provision}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{sk}})$ to $\mathcal{SC}_{\mathsf{B}}$
Encryption Party A:
On message $(\textsc{Setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ from $\mathcal{SC}^{\mathsf{C}}$:
    send GETPK to $G_{\mathsf{att}}$ and receive $\mathsf{vk}_{\mathsf{att}}$
    send GETK to $\mathcal{F}_{\mathsf{CERT}}$ and receive $\mathsf{VK}$
    assert $\Sigma.\mathsf{Vrfy}(\mathsf{vk}, (\mathsf{sid}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}, \mathsf{mpk}), \sigma_{\mathsf{KME}})$
    send GET to $\mathcal{CRS}$ and receive $(\textsc{Crs}, \mathsf{crs})$
    $(\mathsf{vk}_{\Sigma}, \mathsf{sk}_{\Sigma}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$
    send $(\textsc{Sign}, \mathsf{vk}_{\Sigma})$ to $\mathcal{F}_{\mathsf{CERT}}$ and receive cert
    store $\mathsf{mpk}, \mathsf{crs}, \mathsf{vk}_{\Sigma}, \mathsf{sk}_{\Sigma}, \mathsf{cert}$
On message $(\textsc{KeyShareGen}, \mathsf{F}, \mathsf{B})$ from a party $P$:
    $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\Sigma}, \mathsf{F}, \mathsf{B})$
    send $(\textsc{KeyShareGen}, \mathsf{F}, \sigma, \mathsf{vk}_{\Sigma}, \mathsf{cert})$ to $\mathcal{SC}_{B}$
On message $(\textsc{Encrypt}, \mathsf{m}, k)$ from a party $P$:
    assert $\mathsf{mpk} \neq \bot \wedge \mathsf{m} \in \mathcal{X} \wedge k$ is an integer
    $\mathsf{ct} \xleftarrow{r} \mathsf{PKE.Enc}(\mathsf{mpk}, (\mathsf{m}, k))$
    $\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), ((\mathsf{m}, k), \mathsf{r}), \mathsf{crs}), \mathsf{ct}_{\mathsf{msg}} \leftarrow (\mathsf{ct}, \pi)$
    send $(\textsc{write}, \mathsf{ct}_{\mathsf{msg}})$ to $\mathcal{REP}$ and receive h
    return $(\textsc{encrypted}, \mathsf{h})$
Decryption Party B:
On message $(\textsc{Setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ from $\mathcal{SC}^{\mathsf{C}}$:
    $\mathcal{KS} = \{\}, \mathcal{K} = \{\}$
    send GETPK to $G_{\mathsf{att}}$ and receive $\mathsf{vk}_{\mathsf{att}}$
    send GETK to $\mathcal{F}_{\mathsf{CERT}}$ and receive $\mathsf{VK}$
    assert $\Sigma.\mathsf{Vrfy}(\mathsf{vk}, (\mathsf{idx}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}, \mathsf{mpk}), \sigma_{\mathsf{KME}})$
    store $\mathsf{mpk}$; $\mathsf{eid}_{\mathsf{DE}} \leftarrow G_{\mathsf{att}}.\mathsf{install}(\mathsf{B.sid}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}})$
    send GET to $\mathcal{CRS}$ and receive $(\textsc{Crs}, \mathsf{crs})$
    $((\mathsf{pk}_{KD}, \cdot, \cdot), \sigma) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{DE}}, (\mathsf{init\text{-}setup}, \mathsf{eid}_{\mathsf{KME}}, \sigma_{\mathsf{KME}}, \mathsf{crs}, \mathsf{B.sid}))$
    send $(\textsc{provision}, \sigma, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD})$ to $\mathcal{SC}_{\mathsf{C}}$ and receive $(\textsc{provision}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}})$
    $G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{DE}}, (\mathsf{complete\text{-}setup}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}}))$
    return SETUP
On message $(\textsc{KeyShareGen}, \mathsf{F}, \sigma, \mathsf{vk}_{\Sigma}, \mathsf{cert})$ from $\mathcal{SC}^{\mathsf{A}}$:
    if $|\mathcal{KS}[\mathsf{F}]| = 0$ then
        $\mathsf{eid}_{\mathsf{F}} \leftarrow G_{\mathsf{att}}.\mathsf{install}(\mathsf{B.sid}, \mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}[\mathsf{F}]}})$
        $(\mathsf{pk}_{\mathsf{FD}}, \sigma_{\mathsf{F}}) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{F}}, (\mathsf{init}, \mathsf{mpk}, \mathsf{B.sid}))$
        $\mathcal{K}[\mathsf{F}] \leftarrow (\mathsf{eid}_{\mathsf{F}}, \mathsf{pk}_{\mathsf{FD}}, \sigma_{\mathsf{F}})$
    $\mathcal{KS}[\mathsf{F}] \leftarrow \mathcal{KS}[\mathsf{F}] \parallel (\sigma, \mathsf{vk}_{\Sigma}, \mathsf{cert})$
On message $(\textsc{Decrypt}, \mathsf{F}, \mathsf{h})$ from a party $P$:
    assert $\mathcal{K}[\mathsf{F}] \neq \bot$
    send $(\textsc{read}, \mathsf{h})$ to $\mathcal{REP}$ and receive $\mathsf{ct}_{\mathsf{msg}}$
    $(\mathsf{eid}_{\mathsf{F}}, \mathsf{pk}_{\mathsf{FD}}, \sigma_{\mathsf{F}}) \leftarrow \mathcal{K}[\mathsf{F}]$
    $((\mathsf{ct}_{\mathsf{key}}, k_{\mathsf{F}}, \mathsf{crs}), \sigma_{\mathsf{DE}}) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{DE}}, (\mathsf{provision}, \mathcal{KS}[\mathsf{F}], \mathsf{eid}_{\mathsf{F}}, \mathsf{pk}_{\mathsf{FD}}, \sigma_{\mathsf{F}}, \mathsf{F}, \mathsf{B.pid}))$
    $((\mathsf{computed}, \mathsf{y}), \cdot) \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_\mathsf{F}, (\mathsf{run}, \sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{ct}_{\mathsf{key}}, \mathsf{ct}_{\mathsf{msg}}, k_\mathsf{F}, \mathsf{crs}, \perp))$
    return $(\textsc{decrypted}, \mathsf{y})$
  $\underline{\mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}}$
  on input init
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.PGen}(1^\lambda)$
    return pk
  on input $(\mathsf{provision}, (\sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}))$:
    $\mathsf{vk}_{\mathsf{att}} \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{vk}$; fetch crs, idx, sk
    assert $\Sigma.\mathsf{Vrfy}(\mathsf{vk}_{\mathsf{att}}, (\mathsf{idx}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}, (\mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}), \sigma_{\mathsf{DE}})$
    $\mathsf{ct}_{\mathsf{key}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{KD}, \mathsf{sk})$
    return $\mathsf{ct}_{\mathsf{key}}$
```

```
$\underline{\mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}}$
on input $(\mathsf{init\text{-}setup}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}, \mathsf{idx})$:
  assert $\mathsf{pk}_{KD} \neq \perp$
  $(\mathsf{pk}_{KD}, \mathsf{sk}_{KD}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$
  store $\mathsf{sk}_{KD}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}, \mathsf{idx}$
  return $\mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}$
on input $(\mathsf{complete\text{-}setup}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}})$:
  $\mathsf{vk}_{\mathsf{att}} \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{vk}$
  fetch $\mathsf{eid}_{\mathsf{KME}}, \mathsf{sk}_{KD}, \mathsf{idx}$
  $\mathsf{m} \leftarrow (\mathsf{idx}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}, \mathsf{ct}_{\mathsf{key}})$
  assert $\Sigma.\mathsf{Vrfy}(\mathsf{vk}_{\mathsf{att}}, \mathsf{m}, \sigma_{\mathsf{KME}})$
  $\mathsf{sk} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_{KD}, \mathsf{ct}_{\mathsf{key}})$
  store $\mathsf{sk}, \mathsf{vk}_{\mathsf{att}}$
on input $(\mathsf{provision}, \mathcal{KS}_\mathsf{F}, \mathsf{eid}_\mathsf{F}, \mathsf{pk}_{\mathsf{FD}}, \sigma_\mathsf{F}, \mathsf{F}, \mathsf{pid})$:
  fetch $\mathsf{eid}_{\mathsf{KME}}, \mathsf{vk}_{\mathsf{att}}, \mathsf{sk}, \mathsf{idx}, \mathsf{crs}$
  $\mathsf{m} \leftarrow (\mathsf{idx}, \mathsf{eid}, \mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\mathsf{F}]}, \mathsf{pk}_{\mathsf{FD}})$
  assert $\forall (\sigma_{\mathsf{vk}_\cdot}, \mathsf{vk}_\Sigma, \mathsf{cert}) \in \mathcal{KS}_\mathsf{F}:$
  $: (\Sigma.\mathsf{Vrfy}(\mathsf{VK}, \mathsf{vk}_\Sigma, \mathsf{cert}) \wedge \Sigma.\mathsf{Vrfy}(\mathsf{vk}_\Sigma, (\mathsf{F}, \mathsf{pid}), \sigma_{\mathsf{vk}_\cdot}) \wedge$
  $\wedge \forall \mathsf{vk}_\Sigma$ are distinct) $\wedge \Sigma.\mathsf{Vrfy}(\mathsf{vk}_{\mathsf{att}}, \mathsf{m}, \sigma_\mathsf{F})$
  return $\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{FD}}, \mathsf{sk}), |\mathcal{KS}_\mathsf{F}|, \mathsf{crs}$
```

```
$\underline{\mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\mathsf{F}]}}$
on input $(\mathsf{init}, \mathsf{mpk}, \mathsf{idx})$:
  assert $\mathsf{pk}_{\mathsf{FD}} = \perp$
  $(\mathsf{pk}_{\mathsf{FD}}, \mathsf{sk}_{\mathsf{FD}}) = \mathsf{PKE.Gen}(1^\lambda)$
  $\mathsf{mem} \leftarrow \emptyset$; store $\mathsf{sk}_{\mathsf{FD}}, \mathsf{mem}, \mathsf{mpk}, \mathsf{idx}$
  return $\mathsf{pk}_{\mathsf{FD}}$
on input $(\mathsf{run}, \sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{ct}_{\mathsf{key}}, \mathsf{ct}_{\mathsf{msg}}, k_\mathsf{F}, \mathsf{crs}, \mathsf{y}')$:
  if $\mathsf{y}' \neq \perp$
    return $(\mathsf{computed}, \mathsf{y}')$
  $\mathsf{vk}_{\mathsf{att}} \leftarrow \mathcal{G}_{\mathsf{att}}.\mathsf{vk}$; $(\mathsf{ct}, \pi) \leftarrow \mathsf{ct}_{\mathsf{msg}}$
  fetch $\mathsf{sk}_{\mathsf{FD}}, \mathsf{mem}, \mathsf{mpk}, \mathsf{idx}$
  $\mathsf{m} \leftarrow (\mathsf{idx}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}, (\mathsf{ct}_{\mathsf{key}}, k_\mathsf{F}, \mathsf{crs}))$
  assert $\Sigma.\mathsf{Vrfy}(\mathsf{vk}_{\mathsf{att}}, \mathsf{m}, \sigma_{\mathsf{DE}})$
  $\mathsf{sk} = \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{FD}}, \mathsf{ct}_{\mathsf{key}})$
  assert $\mathsf{N}.\mathcal{V}((\mathsf{mpk}, \mathsf{ct}), \pi, \mathsf{crs})$
  $(\mathsf{x}, k) = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$
  assert $k_\mathsf{F} \geq k'$
  $\mathsf{out}, \mathsf{mem}' = \mathsf{F}(\mathsf{x}, \mathsf{mem})$
  store $\mathsf{mem} \leftarrow \mathsf{mem}'$
  return $(\mathsf{computed}, \mathsf{out})$
```

**Simulator (sketch).** The simulator that shows indistinguishability between *DD*-FESR and *DD*-Steel is similar to the Steel's FESR simulator in [12, Theorem 3]. Among notable differences, the simulator now maintains a set of locally generated signature keypairs for all parties in **A**, along with their certificates from the certification functionality $\mathcal{F}_{\mathsf{CERT}}$. When notified that the environment has requested a KeyShareGen for some party A, the simulator produces the equivalent to a functional key share by using the local signing key for A, and sends it to the relevant party B. Requests for decryption are executed honestly if the decryptor provides a sufficient number of functional key shares in possession of honest parties (i.e., parties for which the Corrupt message has not been sent). If a key share has been generated locally by the real-world adversary, the simulator verifies that this key share is a valid signature. If so, it sends a KeyShareGen request to *DD*-FESR on behalf of the corresponding corrupted party. Under the same condition, if the decryption is for a message $(\mathsf{x}, k)$, where there are more than $k$ corrupted parties, then the simulator will allow the decryption of any function on x. All other adversarial behaviour is handled the same way as in Steel to FESR simulator. We omit the proof for brevity. Note that the necessary proof hybrids rely on the unforgeability of signature schemes used for functional key share generation and by the certification functionality $\mathcal{F}_{\mathsf{CERT}}$. We maintain the same identity bound used in the original version, and the preconditions for the UCGS composition theorem holds for our protocols just as they do for the centralised version.

### 3.3 Turning Stateful functions into Multi-Input

As pointed out in [12], FESR subsumes Multi-Input Functional Encryption [25] in that it is possible to use the state to emulate functions over multiple inputs. We now briefly outline how to realise a Multi-Input functionality using FESR (and by extension *DD*-FESR) through the definition of a simple compiler from single-input stateful functions to multi-input functions. To compute a stateless, multi-input function $\mathrm{F} : (\underbrace{\mathcal{X} \times \cdots \times \mathcal{X}}_{n}) \to \mathcal{Y}$ we define the following stateful functionality:

```
function $\mathrm{Agg}_{\mathrm{F},n}(\mathrm{x}, \mathrm{s})$
    if $|\mathrm{s}| < n$ then return $(\perp, \mathrm{s} \parallel \mathrm{x})$
    else return $(\mathrm{F}(\mathrm{s}||\mathrm{x}), \emptyset)$        ▷ s is equivalent to the array containing $\mathrm{x}_1, \ldots, \mathrm{x}_{n-1}$
```

where input $x$ is in $\mathcal{X}$, the state s is in $\mathcal{S}$, and $n$ is bounded by the maximum size of $\mathcal{S}$.

The above aggregator function is able to merge the inputs of multiple encryptors because in FESR the state of a function is distinct between each decryptor; therefore, multiple decryptors attempting to aggregate inputs will not interfere with each other's functions. Note that in the above formulation the order of parameters relies on the decryptor's sequence of invocations. If F is a function where the order of inputs affects the result, malicious decryptor B could choose not to run decryption in the same order of inputs as received. It is possible to further extend the decryption function to respect the order of parameters set by each encryptor. If a subset of encryptors is malicious, we can parametrise the function by a set of public keys for each party, and ask them to sign their inputs. Furthermore, the compiler can be easily extended to multi-input *stateful* functionalities, by keeping the list of inputs as a variable within the state array and not discarding the state on the $n$-th invocation of the compiler. One additional advantage of implementing Multiple-Input functionalities through stateful functionalities is that we are not constrained to functions with a fixed number of inputs. If we treat the inner functionality to the compiler as a function taking as input a list, we can use the same compiler functionality for inner functions of any $n$-arity. We denote this type of functions as $[\![\mathcal{X}]\!] \to \mathcal{Y}$.

## 4 Glass-Vault PLATFORM

In this section, we present the Glass-Vault platform. First, we provide the formal definition of "Analysis-augmented Exposure Notification" ($\mathrm{EN}^+$) which is an extension of the standard Exposure Notification (EN), proposed in [19], to allow arbitrary computation on data shared by users. Then, we present Glass-Vault and show that it UC-realises the ideal functionality of $\mathrm{EN}^+$, $\mathcal{F}_{\mathrm{EN}^+}$.

### 4.1 Analysis-augmented Exposure Notification ($\mathrm{EN}^+$)

Since $\mathrm{EN}^+$ is built upon EN, we first re-state relevant notions used in the UC modelling of EN. Namely, EN relies on the time functionality $\mathbb{T}$ and the "physical reality" functionality $\mathbb{R}$ that we present in Appendix B.6 and B.7, respectively. In particular, $\mathbb{R}$ models the occurrence of events in the physical world (e.g., users' motion or location data). Measurements of a real-world event are sent as input from the environment, and each party can retrieve a list of their own measurements. Using $\mathbb{T}$ and $\mathbb{R}$ as subroutines, the functionality of EN, $\mathcal{F}_{\mathsf{EN}}$, presented formally in Appendix B.8, is defined in terms of a risk estimation function $\rho$, a leakage function $\mathcal{L}$, a set of allowable measurement error functions $E$, and a set of allowable faking functions $\Phi$. The functionality queries $\mathbb{R}$ and applies the measurement error function chosen by the simulator to compute a "noisy record of reality". This in turn is used to decide whether to mark a user as infected, and to compute a risk estimation score for any user. The adversary can mark some parties as corrupted and obtain leakage of their local state, as well as modifying the physical reality record with a reality-faking function. This allows simulation of adversarial behaviour such as relay attacks or refusing to notify the protocol of a user's infectiousness. The functionality captures a variety of contact tracing protocols and attacker models through its parameters. For simplicity, it does not model the testing process the users engage in to find out they are positive, and it assumes that once a user is notified of exposure they are removed from the protocol.

The extension to $\mathrm{EN}^+$ involves the addition of an additional entity to the above scheme. Namely, an analyst Ä, which wants to learn a certain function, $\alpha$, on data contributed by exposed users, some of which might be sensitive. The users are thus provided with a mechanism to accept whether an analyst is allowed to receive the result of the executions of any particular function. In order to receive the result, the analyst needs to be authorised by a portion of exposed users determined by function $K$. We denote by SEC a field in the physical reality record for a user to be used for storing any sensitive data. We present a formal definition of $\mathrm{EN}^+$'s ideal functionality, $\mathcal{F}_{\mathrm{EN}^+}$, below.

**Functionality** $\mathcal{F}_{\mathrm{EN}^+}[\rho, E, \Phi, \mathcal{L}, AF, K]$

The functionality is parametrised by exposure risk function $\rho$, physical reality error $E$, faking functions $\Phi$ for misrepresenting the physical reality, and leakage functions $\mathcal{L}$, as in the regular $\mathcal{F}_{\mathrm{EN}}$. $AF$ is the set of all functions $\{\alpha \mid \alpha : [\![\mathcal{X}]\!] \times \mathcal{S} \times \mathcal{R} \rightarrow \mathcal{Y} \times \mathcal{S}\}$ an analyst could be authorised to compute. $K()$ is a function of the current number of users required to determine the minimum threshold of analyst authorisations.

| State variables | Description |
|---|---|
| SE | List of users who have shared their exposure status and time of upload |
| $\overline{U}$ | List of active users; $\mathsf{SE} \cap \overline{U} = \emptyset$ |
| $\widetilde{U}$ | List of corrupted users |
| $\overline{A}$ | For each pair of analyst and allowed function, the dictionary $\overline{A}$ contains the number of users that have authorised this pair |

```
On message (Setup, $\epsilon^*$) from $\mathcal{A}$:
    assert $\epsilon^* \in E$
    $\widetilde{\mathrm{R}}_\epsilon \leftarrow \emptyset$                ▷ Initialise noisy record of physical reality
On message (ActivateMobileUser, $U$) from a party $P$:
    $\overline{U} \leftarrow \overline{U} \parallel U$
    send (ActivateMobileUser, $U$) to $\mathcal{A}$
On message (ShareExposure, $U$) from a party $P$:
    send (AllMeas, $\epsilon^*$) to $\mathbb{R}$ and receive $\widetilde{\mathrm{R}}^*$
    $\widetilde{\mathrm{R}}_\epsilon \leftarrow \widetilde{\mathrm{R}}_\epsilon \parallel \widetilde{\mathrm{R}}^*$
    if $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{INFECTED}] = \bot$ then return error
    else
        $\mu \leftarrow \widetilde{\mathrm{R}}_\epsilon[U] \parallel \widetilde{\mathrm{R}}_\epsilon[\mathsf{SE}]$
        send time to $\mathbb{T}$ and receive $t$
        $\mathsf{SE} \leftarrow \mathsf{SE} \parallel (U, t); \overline{U} \leftarrow \overline{U} \setminus \{U\}$
        if $U \in \widetilde{U}$ then send (ShareExposure, $U$) to $\mathcal{A}$
```

```
On message ($\textsc{ExposureCheck}, U$) from a party $P$:
  if $U \in \overline{U}$ then
      send ($\textsc{AllMeas}, \epsilon^*$) to $\mathbb{R}$ and receive $\widetilde{\mathrm{R}}^*$
      $\widetilde{\mathrm{R}}_\epsilon \leftarrow \widetilde{\mathrm{R}}_\epsilon \parallel \widetilde{\mathrm{R}}^*; \mu \leftarrow \widetilde{\mathrm{R}}_\epsilon[U] \parallel \widetilde{\mathrm{R}}_\epsilon[\mathrm{SE}]$
      return $\rho(U, \mu)$
  else return error
On message ($\textsc{Corrupt}, U$) from $\mathcal{A}$:
  $\widetilde{U} \leftarrow \widetilde{U} \parallel U$
On message ($\textsc{RegisterAnalyst}$, Ä, $\alpha$) from a party $P$:
  if $\alpha \in AF$ then
      for all $U \in \mathrm{SE}$ do send ($\textsc{RegisterAnalystAccept}$, Ä, $\alpha$) to $U$
      $\overline{A}[\text{Ä}, \alpha] \leftarrow 0$
  send ($\textsc{RegisterAnalyst}, \alpha$, Ä) to $\mathcal{A}$
On message ($\textsc{RegisterAnalystAccept}, \alpha$, Ä, $U$) from a party $P$:
  if $U \in \widetilde{U}$ then send ($\textsc{RegisterAnalystAccept}, \alpha$, Ä, $U$) to $\mathcal{A}$ and await OK; then
  $\overline{A}[\text{Ä}, \alpha] \leftarrow \overline{A}[\text{Ä}, \alpha]{+}{+}$
  send ($\textsc{RegisterAnalystAccept}, U, \alpha$) to Ä
On message ($\textsc{RemoveMobileUser}, U$) from a party $P$:
  $\overline{U} \leftarrow \overline{U} \setminus \{U\}$
On message ($\textsc{Analyse}, \alpha$) from a party $P$:
  if $\overline{A}[P, \alpha] \geq K(|\mathrm{SE}|)$ then
      $y \leftarrow \alpha(\widetilde{\mathrm{R}}_\epsilon[\mathrm{SE}][\mathrm{SEC}])$
      send ($\textsc{Analysed}, \alpha, P, y$) to $\mathcal{A}$
      return $y$
On message ($\textsc{MyCurrentMeas}, U, A, e$) from $\mathcal{A}$:
  if $U \in \widetilde{U}$ then
      send ($\textsc{MyCurrentMeas}, U, A, e$) to $\mathbb{R}$ and receive $u_A^e$
      send ($\textsc{MyCurrentMeas}, u_A^e$) to $\mathcal{A}$
On message ($\textsc{FakeReality}, \phi$) from $\mathcal{A}$:
  if $\phi \in \Phi$ then $\widetilde{\mathrm{R}}_\epsilon \leftarrow \phi(\widetilde{\mathrm{R}}_\epsilon)$
On message $\textsc{Leak}$ from $\mathcal{A}$:
  send ($\textsc{Leak}, \mathcal{L}(\{\widetilde{\mathrm{R}}_\epsilon, \overline{U}, \mathrm{SE}\})$) to $\mathcal{A}$
On message ($\textsc{IsCorrupt}, U$) from $\mathcal{Z}$:
  return $U \overset{?}{\in} \widetilde{U}$
```

### 4.2 Glass-Vault **protocol**

In this section, we present the Glass-Vault protocol. It is a delicate combination of two primary primitives; namely, (i) the original exposure notification proposed by Canetti et al. [19], and (ii) the enhanced functional encryption that we proposed in Section 3. In this protocol, infected users upload not only the regular data needed for exposure notification but also the encryption of their sensitive measurements (e.g., their GPS coordinates, electronic health records, environment's air quality). Once there is a need to perform data analytics by executing specific computations on users' outsourced data, the users are informed via public announcements. At this stage, users can provide (functional keys as) permission tokens to the analyst, who can run such computations only if the number of tokens exceeds a threshold defined by the function $K$ over the number of parties in the set $\mathbf{A}$ of $DD$-FESR (note that since a party is registered to $DD$-FESR only if they are exposed, the number of exposed users matches the size of $\mathbf{A}$). Due to the security of the proposed functional encryption, the Glass-Vault auditor does not learn anything about the users' sensitive inputs beyond what the function evaluation reveals.

It is not hard to see that this protocol (a) is generic, as it supports arbitrary secure computations (i.e., multi-input stateful and randomised functions) on users' shared data, and (b) is transparent, as computations are performed only if permission is granted by a sufficient number of users and in that the user can choose whether they are willing to share their sensitive data or not, making the collection of information consensual.

Besides the $DD$-FESR functionality, the Glass-Vault protocol makes use of the physical reality functionality $\mathbb{R}$, the exposure notification functionality $\mathcal{F}_{\mathsf{EN}}$, and the trusted bulletin board functionality $\mathcal{F}_{\mathsf{TBB}}$, described in Appendix B.7, B.8, and B.9, respectively.

**Protocol** Glass-Vault$[\rho, E, \Phi, \mathcal{L}, AF, K, \mathsf{C}]$

The protocol takes the same types of parameters as defined in $\mathcal{F}_{\mathsf{EN^+}}$, and the identity of a trusted authority C. We use $U$ to refer to a normal user of the Exposure Notification System (corresponding to A in $DD$-FESR). We use Ä to refer to an analyst (corresponding to $DD$-FESR's decryptor, B). Among other ideal setups, Glass-Vault leverages the exposure notification ideal functionality $\mathsf{EN}[\rho, E, \Phi, \mathcal{L}]$ and functional encryption ideal functionality $DD$-FESR$[AF, \emptyset, \emptyset, \mathsf{C}]$.

```
User $U$ :
On message ActivateMobileUser from a party $P$:
  send (ActivateMobileUser, $U$) to $\mathcal{F}_{\mathsf{EN}}$
On message ShareExposure from a party $P$:
  send (ShareExposure, $U$) to $\mathcal{F}_{\mathsf{EN}}$ and receive $r$
  if $r \neq$ error then
    send Setup to $DD$-FESR
    send (MyCurrentMeas, $U$, SEC, $e$) to $\mathbb{R}$ and receive $u^e_{\mathsf{SEC}}$
    send (Encrypt, $u^e_{\mathsf{SEC}}$, $K(|DD\text{-FESR}.\mathbf{A}|)$) to $DD$-FESR and receive (encrypted, h)
    erase $u^e_{\mathsf{SEC}}$ and send (Add, h) to $\mathcal{F}_{\mathsf{TBB}}$
On message ExposureCheck from a party $P$:
  send (ExposureCheck, $U$) to $\mathcal{F}_{\mathsf{EN}}$ and receive $\rho_U$
  if $\rho_U \neq$ error then return $\rho_U$
On message (RegisterAnalystAccept, $\alpha$, Ä) from a party $P$:
  send (KeyShareGen, $\alpha$, Ä) to $DD$-FESR
  send (RegisterAnalystAccept, $U$, $\alpha$) to Ä
Analyst Ä:
On message (RegisterAnalyst, $\alpha$, Ä) from a party $P$:
  send Setup to $DD$-FESR
  for all exposed $U$ do send (RegisterAnalystAccept, $\alpha$, Ä) to $U$
```

```
On message $(\textsc{Analyse}, \alpha)$ from a party $P$:
  send Retrieve to $\mathcal{F}_{\mathrm{TBB}}$ and receive $C$
  for $\mathsf{h} \in C$ do
    send $(\textsc{Decrypt}, h, \alpha)$ to $DD$-FESR and receive $(\textsc{decrypted}, \mathrm{y})$
    if $\mathrm{y} \neq \bot$ then return $(\textsc{Decrypted}, \alpha, P, \mathrm{y})$
```

Theorem 4.1. *Let $\rho, E, \Phi, \Phi^+, \mathcal{L}, \mathcal{L}^+, AF, K, \mathrm{C}$ be parameters such that the following conditions hold: (1) $\Phi \subset \Phi^+$, (2) for every input $x$, $\mathcal{L}(x) \subset \mathcal{L}^+(x)$, and (3) there is a function $\phi^+ \in \Phi^+$ such that for every input $x$, every noisy physical reality record $\widetilde{\mathrm{R}}_\epsilon \in x$, every function $\phi \in \Phi$, and every user or set of users $U$, (3.1) $\mathcal{L}(x)$ does not contain any instruction to leak the contents of sensitive data record $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{SEC}]$ but $\mathcal{L}^+(x)$ does, and (3.2) $\phi(\widetilde{\mathrm{R}}_\epsilon)$ does not tamper with record $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{SEC}]$ but $\phi^+(\widetilde{\mathrm{R}}_\epsilon)$ does.*

*Then, the* Glass-Vault *protocol is secure, i.e., it holds that*

$$\text{Glass-Vault}[\rho, E, \Phi, \mathcal{L}, AF, K, \mathrm{C}] \text{ UC-realises } \mathcal{F}_{\mathrm{EN}^+}[\rho, E, \Phi^+, \mathcal{L}^+, AF, K],$$

*in the presence of global functionalities $\mathbb{T}$ and $\mathbb{R}$.*

Proof (sketch). The high-level task of our simulator is to *synchronise* the inputs of the $\alpha$ functions between the ideal world (where they are stored in $\mathbb{R}$), and the real world (where they are held in the $DD$-FESR ideal repository) There are two directions for which we need to provide simulation: (1) when honest users share their exposure status and related sensitive data; (2) when a malicious user encrypts via $DD$-FESR dishonestly generated data (in that they do not match with the corrupted user's physical reality measurements). Our simulator addresses point (1) by using the leakage function $\mathcal{L}^+$ to learn the honest user's sensitive data. It addresses point (2) by using functions in $\Phi^+$ to modify the noisy record of physical reality in $\mathcal{F}_{\mathrm{EN}^+}$ to match with the malicious inputs. As the simulator keeps the states synchronised between real world and ideal world, when any registered and corrupted analyst executes an Analyse request in the ideal world, the simulator can send messages on behalf of the analyst to run the corresponding $DD$-FESR decryption subroutine in the real world. Likewise, if the adversary attempts to decrypt through $DD$-FESR directly, the simulator will trigger the ideal Analyse subroutine, after having modified the physical reality records appropriately to match the decryption's inputs. We also simulate the RegisterAnalyst and RegisterAnalystAccept sequence of operations by triggering the corresponding Setup and KeyShareGen subroutines in $DD$-FESR. Any other adversarial calls to $\mathcal{F}_{\mathrm{EN}^+}$ such as $(\textsc{Setup}, \epsilon^*)$ and $(\textsc{FakeReality}, \phi)$ are allowed and redirected to $\mathcal{F}_{\mathrm{EN}}$, as long as $\epsilon^* \in E$ and $\phi \in \Phi \wedge \phi \notin (\Phi^+ \setminus \Phi)$. □

Cost-wise, in the protocol, an infected user's computation and communication complexity for the proposed data analytics purposes is independent of the total number of users, while it is linear with the number of functions requested by the analysts. The analysts' computation overhead depends on each function's complexity and the number of decryptions (as each decryption updates the function's state). The cost to non-infected users is comparable to the best EN protocols that realise $\mathcal{F}_{\mathrm{EN}}$ (such as the protocol in [19, Section 8.5]), as they do not need to run any other operations related to the data analytics stage, besides passively collecting measurements of sensitive data.

## 5 EXAMPLE: INFECTIONS HEATMAP

In this section, we provide a concrete example of a computation that a Glass-Vault analyst might perform: generating a daily heatmap of the current clusters of infections. This is an interesting application of Glass-Vault, as it relies on collecting highly-sensitive location information from infected individuals.

$\mathsf{Heatmap}_{k,q}(\mathsf{x}, \mathsf{s})$ is a multi-input stateful function, parametrised by $k$, the number of distinct cells we divide the map into, and $q$, the minimum number of exposed users that have shared their data. The values of these parameters affect the granularity of the results, computational costs, and privacy of the exposed users. Thus, they need to be approved as part of the KeyShareGen procedure (in that the parameters' values are hardcoded in the Heatmap program, so that different parameter values require different functional keys). Each input to the function $u \in \mathsf{x}$ is a $\mathcal{T} \times k$ matrix, where $\mathcal{T}$ is the incubation period of the virus in days (the number of days for which someone is infectious). Each entry in $u$ contains the number of hours within a day an infected individual spent in a particular location. Location data are collected once every hour by the user's phone, and divided into $k$ bins. Once a user finds out they are positive and triggers ShareExposure, they build the matrix by concatenating their location history for the last $\mathcal{T}$ days (the maximum number of days since they might have been spreading the virus), with the last row of the matrix corresponding to the locations during the most recent day and each row above in decreasing order until the first row which contains the locations $\mathcal{T}$ days ago. Heatmap maintains as part of its state a list, $\mathsf{m}$, of $\mathcal{T}$-sized circular buffers (a FIFO data structure of size $\mathcal{T}$; once more than $\mathcal{T}$ entries have been filled, the buffer starts overwriting data starting from the oldest entry) . On a call with input $\mathsf{x}$, the function allocates a new circular buffer $\mathsf{b}$ for each matrix $u \in \mathsf{x}$, and assigns each of $u$'s rows to one of $\mathsf{b}$'s $\mathcal{T}$ elements, starting from the top row. Each element in $\mathsf{b}$ now contains a list of $k$ geolocations for a specific day, with the first element containing the locations $\mathcal{T}$ days ago, and so on. For any buffer already in $\mathsf{m}$, we append a new zero vector, effectively erasing the record of that user's location for the earliest day. If there is a buffer that is completely zeroed out by this operation, we remove it from $\mathsf{m}$.

If we have $|\mathsf{m}| \geq q$, we return the row-wise sum of vectors $\sum_{\mathsf{b} \in \mathsf{m}} \sum_{i=0}^{\mathcal{T}-1} \mathsf{b}[i]$. The result is a single $k$-sized vector containing the total number of hours spent by all users within the last $\mathcal{T}$ days: our heatmap. The full pseudocode for the function is provided in Appendix C. For the results' correctness, an analyst should run the function (through a decryption operation) once a day. As the summation of user location vectors is a destructive operation, the probability that a malicious analyst can recover any specific user's input will be inversely proportional to $q$.

A common problem with both centralised and decentralised contact tracing systems is that a user can tamper with their own client applications to upload malicious data (terrorist attack [39]). Since there is no secure pipeline from the raw measurements from sensors to a specific application, unless we adopt the strong requirement that every client also runs a TEE (as in [27]), it is impossible to certify that the users' inputs are valid. Like most other remote computation systems, Glass-Vault cannot provide a blanket protection against this kind of attack. However, due its generality, Glass-Vault allows analysts to use functions that include "sanity checks" to ensure that the data being uploaded are at least sensible, in order to limit the damage that the attack may cause. In the heatmap case, one such check could be verifying that for each row of $u$, it must hold that its column-wise sum is equal to 24, since each row represents the number of hours spent across various locations by the user in a day (assuming the user's phone is on and able to collect their location at least once an hour throughout a day). To capture this type of attack in the ideal functionality $\mathcal{F}_{\mathrm{EN}^+}$, we instantiate it with a FakeReality function in $\Phi^+$ such that, if a malicious user $U$ uploads this type of fake geolocation, it will update $U$'s position within the noisy record of physical reality to match $U$'s claimed location, while making sure that other users who compute risk exposure and have been in close contact with $U$ will still be notified.

We highlight that Bruni et al. [15] propose an ad-hoc scheme that offers a similar operation. Their scheme relies on combining infection data provided by health authorities with the mass collection of cellphone location data from mobile phone operators. Unlike Glass-Vault that enjoys a strong level of transparency, the approach in [15] does not support any mechanism that allows the subjects of data collection to provide their direct consent.

## 6 FUTURE DIRECTIONS

There are several possible directions for future research. An immediate goal would be to implement Glass-Vault for relevant types of data analytics workloads, examine its run-time, and optimise the system's bottlenecks. Furthermore, it might be desirable to extend the platform to provide a way for external stakeholders to verify the output of analytics function, which is an appealing option if the results can influence public policy. It is also interesting to investigate techniques for malformed inputs detection and fine-grained authorisation policies for approving analysis functions.

## REFERENCES

[1] Roba Abbas and Katina Michael. 2020. COVID-19 contact trace app deployments: learnings from Australia and Singapore. *IEEE Consumer Electronics Magazine* 9, 5 (2020), 65–70.

[2] William Abramson, William J. Buchanan, Sarwar Sayeed, Nikolaos Pitropakis, and Owen Lo. 2021. PAN-DOMAIN: Privacy-preserving Sharing and Auditing of Infection Identifier Matching. *CoRR* abs/2112.02855 (2021). arXiv:2112.02855 https://arxiv.org/abs/2112.02855

[3] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. 2021. Multi-Party Functional Encryption. In *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13043)*, Kobbi Nissim and Brent Waters (Eds.). Springer, 224–255. https://doi.org/10.1007/978-3-030-90453-1_8

[4] Nadeem Ahmed, Regio A. Michelin, Wanli Xue, Sushmita Ruj, Robert A. Malaney, Salil S. Kanhere, Aruna Seneviratne, Wen Hu, Helge Janicke, and Sanjay K. Jha. 2020. A Survey of COVID-19 Contact Tracing Apps. *IEEE Access* 8 (2020), 134577–134601. https://doi.org/10.1109/ACCESS.2020.3010226

[5] Fraunhofer AISEC. 2020. Pandemic Contact Tracing Apps: DP-3T, PEPP-PT NTK, and ROBERT from a Privacy Perspective. Cryptology ePrint Archive, Report 2020/489. https://eprint.iacr.org/2020/489.

[6] Hannah Alsdurf, Yoshua Bengio, Tristan Deleu, Prateek Gupta, Daphne Ippolito, Richard Janda, Max Jarvie, Tyler Kolody, Sekoul Krastev, Tegan Maharaj, Robert Obryk, Dan Pilat, Valerie Pisano, Benjamin Prud'homme, Meng Qu, Nasim Rahaman, Irina Rish, Jean-Franois Rousseau, Abhinav Sharma, Brooke Struck, Jian Tang, Martin Weiss, and Yun William Yu. 2020. Covi White Paper. *CoRR* (2020). arXiv:2005.08502 [cs.CR] http://arxiv.org/abs/2005.08502v1

[7] Tehilla Shwartz Altshuler and Rachel Aridor Hershkowitz. 2020. How Isreal's COVID-19 mass surveillance operation works. https://www.brookings.edu/techstream/how-israels-covid-19-mass-surveillance-operation-works/.

[8] Apple and Google. 2020. Exposure Notification API. https://www.google.com/covid19/exposurenotifications/.

[9] Gennaro Avitabile, Vincenzo Botta, Vincenzo Iovino, and Ivan Visconti. 2020. Towards Defeating Mass Surveillance and SARS-CoV-2: The Pronto-C2 Fully Decentralized Automatic Contact Tracing System. Cryptology ePrint Archive, Report 2020/493. https://eprint.iacr.org/2020/493.

[10] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. 2020. Universal Composition with Global Subroutines: Capturing Global Setup Within Plain UC. 1–30. https://doi.org/10.1007/978-3-030-64381-2_1

[11] Saikrishna Badrinarayanan, Vipul Goyal, Aayush Jain, and Amit Sahai. 2016. Verifiable Functional Encryption. 557–587. https://doi.org/10.1007/978-3-662-53890-6_19

[12] Pramod Bhatotia, Markulf Kohlweiss, Lorenzo Martinico, and Yiannis Tselekounis. 2021. Steel: Composable Hardware-Based Stateful and Randomised Functional Encryption. 709–736. https://doi.org/10.1007/978-3-030-75248-4_25

[13] Jean-François Biasse, Sriram Chellappan, Sherzod Kariev, Noyem Khan, Lynette Menezes, Efe Seyitoglu, Charurut Somboonwit, and Attila Yavuz. 2020. Trace-Σ: a privacy-preserving contact tracing app. Cryptology ePrint Archive, Report 2020/792. https://eprint.iacr.org/2020/792.

[14] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional Encryption: Definitions and Challenges. 253–273. https://doi.org/10.1007/978-3-642-19571-6_16

[15] Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. 2020. Privately Connecting Mobility to Infectious Diseases via Applied Cryptography. Cryptology ePrint Archive, Report 2020/522. https://eprint.iacr.org/2020/522.

[16] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. 136–145. https://doi.org/10.1109/SFCS.2001.959888

[17] Ran Canetti. 2004. Universally Composable Signature, Certification, and Authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 219. https://doi.org/10.1109/CSFW.2004.24

[18] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. 2007. Universally Composable Security with Global Setup. 61–85. https://doi.org/10.1007/978-3-540-70936-7_4

[19] Ran Canetti, Yael Tauman Kalai, Anna Lysyanskaya, Ronald L. Rivest, Adi Shamir, Emily Shen, Ari Trachtenberg, Mayank Varia, and Daniel J. Weitzner. 2020. Privacy-Preserving Automated Exposure Notification. Cryptology ePrint Archive, Report 2020/863. https://eprint.iacr.org/2020/863.

[20] Ran Canetti, Ari Trachtenberg, and Mayank Varia. 2020. Anonymous Collocation Discovery: Harnessing Privacy to Tame the Coronavirus. arXiv:2003.13670 [cs.CY]

[21] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. 2020. DESIRE: A Third Way for a European Exposure Notification System Leveraging the Best of Centralized and Decentralized Systems. *CoRR* abs/2008.01621 (2020). arXiv:2008.01621 https://arxiv.org/abs/2008.01621

[22] Centers for Disease Control and Prevention. 2021. Contact Tracing. https://www.cdc.gov/coronavirus/2019-ncov/daily-life-coping/contact-tracing.html.

[23] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. 2018. Decentralized Multi-Client Functional Encryption for Inner Product. 703–732. https://doi.org/10.1007/978-3-030-03329-3_24

[24] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. 2020. Dynamic Decentralized Functional Encryption. 747–775. https://doi.org/10.1007/978-3-030-56784-2_25

[25] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. 2014. Multi-input Functional Encryption. 578–602. https://doi.org/10.1007/978-3-642-55220-5_32

[26] Yaron Gvili. 2020. Security Analysis of the COVID-19 Contact Tracing Specifications by Apple Inc. and Google Inc. Cryptology ePrint Archive, Report 2020/428. https://eprint.iacr.org/2020/428.

[27] Daniel Günther, Marco Holz, Benjamin Judkewitz, Helen Möllering, Benny Pinkas, and Thomas Schneider. 2020. PEM: Privacy-preserving Epidemiological Modeling. Cryptology ePrint Archive, Report 2020/1546. https://ia.cr/2020/1546.

[28] Wouter Lueks, Seda F. Gürses, Michael Veale, Edouard Bugnion, Marcel Salathé, Kenneth G. Paterson, and Carmela Troncoso. 2021. CrowdNotifier: Decentralized Privacy-Preserving Presence Tracing. 2021, 4 (Oct. 2021), 350–368. https://doi.org/10.2478/popets-2021-0074

[29] Vikram Sharma Mailthody, James Wei, Nicholas Chen, Mohammad Behnia, Ruihao Yao, Qihao Wang, Vedant Agrawal, Churan He, Lijian Wang, Leihao Chen, Amit Agarwal, Edward Richter, Wen-Mei Hwu, Christopher W. Fletcher, Jinjun Xiong, Andrew Miller, and Sanjay Patel. 2021. Safer Illinois and Rokwall: Privacy Preserving University Health Apps for Covid-19. *CoRR* (2021). arXiv:2101.07897 [cs.CR] http://arxiv.org/abs/2101.07897v1

[30] Tania Martin, Georgios Karopoulos, José L. Hernández-Ramos, Georgios Kambourakis, and Igor Nai Fovino. 2020. Demystifying Covid-19 Digital Contact Tracing: a Survey on Frameworks and Mobile Apps. *CoRR* (2020). arXiv:2007.11687 [cs.CR] http://arxiv.org/abs/2007.11687v1

[31] Carla Mascia, Massimiliano Sala, and Irene Villa. 2021. A survey on Functional Encryption. *CoRR* abs/2106.06306 (2021). arXiv:2106.06306 https://arxiv.org/abs/2106.06306

[32] Christian Matt and Ueli Maurer. 2013. A Definitional Framework for Functional Encryption. Cryptology ePrint Archive, Report 2013/559. https://eprint.iacr.org/2013/559.

[33] Rafael Pass, Elaine Shi, and Florian Tramèr. 2017. Formal Abstractions for Attested Execution Secure Processors. 260–289. https://doi.org/10.1007/978-3-319-56620-7_10

[34] Leonie Reichert, Samuel Brack, and Björn Scheuermann. 2020. Ovid: Message-based Automatic Contact Tracing. Cryptology ePrint Archive, Report 2020/1462. https://eprint.iacr.org/2020/1462.

[35] David Sturzenegger, Aetienne Sardon, Stefan Deml, and Thomas Hardjono. 2020. Confidential Computing for Privacy-Preserving Contact Tracing. *CoRR* (2020). arXiv:2006.14235 [cs.CY] http://arxiv.org/abs/2006.14235v1

[36] Florian Tramer, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. 2016. Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge. Cryptology ePrint Archive, Report 2016/635. https://eprint.iacr.org/2016/635.

[37] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. 2020. Epione: Lightweight Contact Tracing with Strong Privacy. *IEEE Data Eng. Bull.* 43, 2 (2020), 95–107. http://sites.computer.org/debull/A20june/p95.pdf

[38] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James R. Larus, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth G. Paterson, Srdjan Capkun, David A. Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel P. Smart, Aysajan Abidin, Seda Gurses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. 2020. Decentralized Privacy-Preserving Proximity Tracing. *IEEE Data Eng. Bull.* 43, 2 (2020), 36–66. http://sites.computer.org/debull/A20june/p36.pdf

[39] Serge Vaudenay. 2020. Centralized or Decentralized? The Contact Tracing Dilemma. Cryptology ePrint Archive, Report 2020/531. https://eprint.iacr.org/2020/531.

[40] Serge Vaudenay and Martin Vuagnoux. 2020. The Dark Side of SwissCovid. https://lasec.epfl.ch/people/vaudenay/swisscovid.

[41] Pengfei Wu, Qingni Shen, Robert H. Deng, Ximeng Liu, Yinghui Zhang, and Zhonghai Wu. 2019. ObliDC: An SGX-based Oblivious Distributed Computing Framework with Formal Proof. 86–99. https://doi.org/10.1145/3321705.3329822

## A SYNTAX OF FUNCTIONAL ENCRYPTION

FE is defined over a class of functions $\mathsf{F} = \{\mathsf{F} \mid \mathsf{F} : \mathcal{X} \to \mathcal{Y}\}$, where $\mathcal{X}$ is the domain and $\mathcal{Y}$ is the range, consisting of the following algorithms:

- $\underline{\mathsf{Setup}}$ (run by $\mathsf{C} \in \mathbf{C}$). It takes a security parameter $1^\lambda$ as input and outputs a master keypair $(\mathsf{mpk}, \mathsf{msk})$.
- $\underline{\mathsf{KeyGen}}$ (run by $\mathsf{C} \in \mathbf{C}$). It takes $\mathsf{msk}$ and a function's description $\mathsf{F} \in \mathsf{F}$ as inputs and outputs functional key $\mathsf{sk}_\mathsf{F}$.
- $\underline{\mathsf{Enc}}$ (run by $\mathsf{A} \in \mathbf{A}$). It takes a plaintext string $\mathsf{x} \in \mathcal{X}$ and $\mathsf{mpk}$ as inputs. It returns a ciphertext $\mathsf{ct}$ or an error.
- $\underline{\mathsf{Dec}}$ (run by $\mathsf{B} \in \mathbf{B}$). It takes ciphertext $\mathsf{ct}$ and functional key $\mathsf{sk}_\mathsf{F}$ as inputs and returns a value $\mathsf{y} \in \mathcal{Y}$.

Informally, correctly evaluating the decryption operation on a ciphertext $\mathsf{ct} \longleftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{x})$ using functional key $\mathsf{sk}_\mathsf{F}$ should result in $\mathsf{y} \longleftarrow \mathsf{F}(\mathsf{x})$. Besides any information that $\mathsf{y}$ reveals about $\mathsf{x}$, the party $\mathsf{B}$ should not learn anything about $\mathsf{A}$'s input, except for some natural leakage from the ciphertext (e.g. the length of the ciphertext).

## B BACKGROUND FUNCTIONALITIES

In this section, we provide an overview of the functionalities that are invoked in the description of $DD$-Steel, $\mathrm{EN}^+$, and Glass-Vault presented in Subsections 3.2, 4.1, and 4.2, respectively. The overview is sufficient for the clarification of the communication interface among the said functionalities and the interacting entities. For a detailed description of each background functionality, we refer readers to the relevant work, except the exposure notification functionality $\mathcal{F}_{\mathsf{EN}}$ that we present in detail to allow an easy comparison with the extended functionality $\mathcal{F}_{\mathsf{EN}^+}$.

### B.1 The global attestation functionality $G_{\mathsf{att}}$

The ideal functionality $G_{\mathsf{att}}$ was introduced in [33] and can be seen as an abstraction for a broad class of attested execution processors. The functionality operates as follows:

- On message INITIALIZE from a party $P$, it generates a pair of signing and verification keys $(\mathsf{spk}, \mathsf{ssk})$. It stores $\mathsf{spk}$ as the master verification key $\mathsf{vk}_{\mathsf{att}}$, available to enclave programs, and $\mathsf{ssk}$ as the master secret key $\mathsf{msk}$, protected by the hardware.
- On message GETPK from a party $P$, it returns $\mathsf{vk}_{\mathsf{att}}$.
- On message (INSTALL, $\mathsf{idx}, \mathsf{prog}$) from a (registered and honest) party $P$, it asserts that $\mathsf{idx} = P.\mathsf{sid}$. Then, it creates a unique enclave identifier $\mathsf{eid}$ and establishes a software enclave for $(\mathsf{eid}, P)$ as $(\mathsf{idx}, \mathsf{prof}, \emptyset)$. It provides $P$ with $\mathsf{eid}$.
- On message (RESUME, $\mathsf{eid}, \mathsf{input}$) from a (registered) party $P$, it calls the enclave $(\mathsf{idx}, \mathsf{prog}, \mathsf{mem})$ for $(\mathsf{eid}, P)$, where $\mathsf{mem}$ is the current memory state. It runs $\mathsf{prog}(\mathsf{input}, \mathsf{mem})$ which returns $\mathsf{output}$ and an update memory state $\mathsf{mem}'$. Finally, it produces a signature $\sigma$ on $(\mathsf{idx}, \mathsf{eid}, \mathsf{prog}, \mathsf{output})$ using $\mathsf{msk}$ and sends $(\mathsf{output}, \sigma)$ to $P$.

### B.2 The certification functionality $\mathcal{F}_{\mathsf{CERT}}$

We assume the existence of an ideal certification functionality $\mathcal{F}_{\mathsf{CERT}}$, inspired by the certification functionality and the certification authority functionality introduced in [17]. The difference between $\mathcal{F}_{\mathsf{CERT}}$ and the certification functionality in [17] is that (i) instead of taking over signature verification, $\mathcal{F}_{\mathsf{CERT}}$ allows the verifier to verify the validity of a signature offline, and (ii) it allows the generation of only one certificate (signature) for each party.

In particular, the functionality $\mathcal{F}_{\mathsf{CERT}}$ exposes methods $GetK$ and $Sign$. On the first call to $GetK$ via a message GETK from a party $P$, it initialises an empty record and generates a signing keypair for signature scheme $\Sigma$, returning the verification key $\mathsf{VK}$ on all subsequent calls to $GetK$. In a call to $Sign$, the input is a message (SIGN, $\mathsf{vk}$) from a party $P$.

The functionality checks that no other message has been recorded from the same UC party id as $P$. If this holds, then it returns the certificate cert, which is a signature on vk under the generated signing key.

### B.3 The common reference string functionality $\mathcal{CRS}$

The $\mathcal{CRS}$ functionality, as described in [12], is parameterised by a distribution $D$. On the first request message GET from a party $P$, the functionality samples a CRS string crs from $D$ and sends crs to $P$. On any subsequent message GET, it returns the same string crs.

### B.4 The secure channel functionality $\mathcal{SC}_R^S$

We adopt the functionality $\mathcal{SC}_R^S$ from [12] that models a secure channel between sender $S$ and receiver $R$. The functionality keeps a record $M$ of the length of messages that are being sent. On message $(\text{SEND}, m)$ from $S$, it sends $(\text{SENT}, m)$ to $R$ and appends the length of $m$ to the record $M$. The adversary is not activated upon sending, but can later on request the record $M$. For simplicity, when the identity of the receiver or the sender is obvious, we will use the notation $\mathcal{SC}^S$ or $\mathcal{SC}_R$, respectively.

### B.5 The repository functionality $\mathcal{REP}$

We relax the functionality $\mathcal{REP}$ from [12] by allowing any party to read/write, as long as the read/write request refers to some specified session. Namely, the functionality keeps a table $M$ of the all the messages submitted by writing requests. On message $(\text{WRITE}, x)$ from $W$, it runs the subroutine `getHandle` to obtain an identifying handle h, and records $x$ in $M[\mathsf{h}]$. On message $(\text{READ}, \mathsf{h})$ from a party $P \in \mathbf{R}$, it returns $M[\mathsf{h}]$ to $P$.

### B.6 The time functionality $\mathbb{T}$

The time functionality $\mathbb{T}$ of [19] can be used as a clock within a UC protocol. It initialises a counter $t$ as 0. On message INCREMENT from the environment, it increments $t$ by 1. On message TIME from a party $P$ , it sends $t$ to $P$.

### B.7 The physical reality functionality $\mathbb{R}$

The functionality $\mathbb{R}$ introduced in [19], represents the "physical reality" of each participant to a protocol, meaning the historical record of all physical facts (e.g., location, motion, visible surroundings) involving the participants.
$\mathbb{R}$ is parameterised by a validation predicate $V$ for checking that the records provided by the environment are sensible, and a set $\mathbf{F}$ of ideal functionalities that have full access to the records obtained by $\mathbb{R}$. The functionality only considers records that have a specific format and include the party identity, time, and the types of measurement (e.g., location, altitude, temperature, distance of the party from each other party, health status) that evaluate the physical reality for the said party. It initialises a list $R$ of all submitted records that are in correct format and operates as follows:

- On message $(P, v)$ from the environment, where $P$ is a party's identity and $v$ is a record in correct format, it appends $(P, v)$ to $R$. Then, it sends TIME to $\mathbb{T}$ (the time functionality presented in B.6) and obtains $t$. It checks that $t$ matches the time entry in $v$ and that $V(R)$ holds. If any check fails, then it halts.
- On message $(\text{MyCurrentMeas}, P, L, e)$ that comes from either party $P$ or a functionality in $\mathbf{F}$ (otherwise, it returns an error), where $L$ is a list of fields that refer to the correct record format and $e$ is an error function:

(1) It finds the latest entry $v$ in the sublist of entries in $R$ whose first element is $P$.
(2) It sets $v_L$ as the record $v$ restricted to the fields in $L$.

(3) It computes $e(v_L)$, i.e., the result of applying the error function $e$ to $v_L$.
(4) It returns $e(v_L)$.

- On message $(\textsc{AllMeas}, e)$ from a functionality in $\mathbf{F}$, it applies $e$ to each record in $R$ and obtains $\tilde{R}$. It returns $\tilde{R}$.

### B.8 The exposure notification functionality $\mathcal{F}_{\mathsf{EN}}$

The Exposure Notification functionality, also introduced in [19], builds on the previous two functionalities to provide a mechanism for warning people who have been exposed to infectious carriers of the virus. The description of the functionality is recapped in section 4.1; we show the formal description for the purposes of comparing this functionality with $\mathcal{F}_{\mathsf{EN}^+}$.

Confirmation of test results when sharing exposure and re-registration into the system for no longer infectious users is not captured by the functionality.

**Functionality $\mathcal{F}_{\mathsf{EN}}[\rho, E, \Phi, \mathcal{L}]$**

| State variables | Description |
|---|---|
| $\widetilde{\mathrm{R}}_\epsilon$ | Noisy record of physical reality |
| $\mathsf{SE}$ | List of users who have shared their exposure status |
| $\overline{U}$ | List of active users |
| $\widetilde{U}$ | List of corrupted users |

```
On message $(\textsc{Setup}, \epsilon^*)$ from $\mathcal{A}$:
  assert $\epsilon^* \in E$;  $\widetilde{\mathrm{R}}_\epsilon \leftarrow \emptyset$
On message $(\textsc{ActivateUser}, U)$ from a party $P$:
  $\overline{U} \leftarrow \overline{U} \parallel U$
  send $(\textsc{ActivateUser}, U)$ to $\mathcal{A}$
On message $(\textsc{RegisterAuditor}, \ddot{\mathrm{A}}, \alpha)$ from a party $P$:
  if $\alpha \in AF$ then
    $\overline{A} \leftarrow \overline{A} \parallel (\ddot{\mathrm{A}}, \alpha)$
  send $(\textsc{RegisterAuditor}, \ddot{\mathrm{A}}, \alpha)$ to $\mathcal{A}$
On message $(\textsc{ShareExposure}, U)$ from a party $P$:
  send $(\textsc{AllMeas}, \epsilon^*)$ to $\mathbb{R}$ and receive $\widetilde{\mathrm{R}}^*$
  $\widetilde{\mathrm{R}}_\epsilon \leftarrow \widetilde{\mathrm{R}}_\epsilon \parallel \widetilde{\mathrm{R}}^*$
  if $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{INFECTED}] = \perp$ then
    return
  else
    $\mu \leftarrow \widetilde{\mathrm{R}}_\epsilon[U] \parallel \widetilde{\mathrm{R}}_\epsilon[\mathsf{SE}]$
    send $\textsc{time}$ to $\mathbb{T}$ and receive $t$
    $\mathsf{SE} \leftarrow \mathsf{SE} \parallel (U, t)$
    $\overline{U} \leftarrow \overline{U} \setminus \{U\}$
    if $U \in \widetilde{U}$ then
      send $\textsc{ShareExposure}$ to $\mathcal{A}$
On message $(\textsc{ExposureCheck}, U)$ from a party $P$:
  if $U \in \overline{U}$ then
```

```
        send (AllMeas, $\epsilon^*$) to $\mathbb{R}$ and receive $\widetilde{\mathrm{R}}^*$
        $\widetilde{\mathrm{R}}_\epsilon \leftarrow \widetilde{\mathrm{R}}_\epsilon \parallel \widetilde{\mathrm{R}}^*$
        $\mu \leftarrow \widetilde{\mathrm{R}}_\epsilon[U] \parallel \widetilde{\mathrm{R}}_\epsilon[\mathrm{SE}]$
        return $\rho(U, \mu)$
    else return error
  On message (RemoveMobileUser, $U$) from a party $P$:
    $\overline{U} \leftarrow \overline{U} \setminus \{U\}$
  On message (Audit, $\alpha$) from a party $P$:
    if $(P, \alpha) \in \overline{A}$ then
        $y \leftarrow \alpha(\widetilde{\mathrm{R}}_\epsilon[\mathrm{SE}][\mathrm{SEC}])$ send (Audit, $\alpha, P, y$) to $\mathcal{A}$
        return $y$
  On message (Corrupt, $U$) from $\mathcal{A}$:
    $\widetilde{U} \leftarrow \widetilde{U} \parallel U$
  On message (MyCurrentMeas, $U, A, e$) from $\mathcal{A}$:
    if $U \in \widetilde{U}$ then
        send (MyCurrentMeas, $U, A, e$) to $\mathbb{R}$ and receive $u_A^e$
        send (MyCurrentMeas, $u_A^e$) to $\mathcal{A}$
  On message (FakeReality, $\phi$) from $\mathcal{A}$:
    if $\phi \in \Phi$ then
        $\widetilde{\mathrm{R}}_\epsilon \leftarrow \phi(\widetilde{\mathrm{R}}_\epsilon)$
  On message Leak from $\mathcal{A}$:
    send (Leak, $\mathcal{L}(\{\widetilde{\mathrm{R}}_\epsilon, \overline{U}, \mathrm{SE}\})$) to $\mathcal{A}$
  On message (IsCorrupt, $U$) from $\mathcal{Z}$:
    return $U \overset{?}{\in} \widetilde{U}$
```

## B.9 The trusted bulletin board functionality $\mathcal{F}_{\mathrm{TBB}}$

The functionality $\mathcal{F}_{\mathrm{TBB}}$, as presented in [19], maintains a state that is updated whenever new data are uploaded (for infectious parties). It initializes a list $C$ of records. On message (Add, $c$) from a party $P$, it checks with $\mathbb{R}$ whether $P$ is infectious (formally, $\mathcal{F}_{\mathrm{TBB}}$ sends a message (MyCurrentMes, $P$, "health_status", id) to $\mathbb{R}$, where id is the identity function). If this holds, then it appends $c$ to $C$. On message Retrieve from a party $P$, it returns $C$ to $P$.

## C HEATMAP PSEUDOCODE

In this section, we provide the pseudocode for the heatmap function discussed in Section 5.

```
function Heatmap$_{k,q}$(x, state)
    if state = $\emptyset$ then m $\leftarrow$ [ ]
    for $c \in$ m do
        $c \leftarrow c \parallel \vec{0}$
        if $\forall i \in c : i = \vec{0}$ then m $\leftarrow$ m $\setminus c$
    for $u \in$ x do
        $b \leftarrow$ CircularBuffer($\mathcal{T}$)
        for $\{i = 0; i < \mathcal{T}; i{+}{+}\}$ do
```

```
            **assert** $\sum_{j=0}^{k-1} u[i, j] = 24$
            $b \leftarrow b \parallel u[i, :]$
        $\mathsf{m} \leftarrow \mathsf{m} \parallel b$
    $\mathsf{y} \leftarrow \vec{0}$
    **if** $|\mathsf{m}| \geq q$ **then**
        **for** $u \in \mathsf{m}$ **do**
            **for** $\{i = 0; i < \mathcal{T}; i{+}{+}\}$ **do**
                $\mathsf{y} \leftarrow \mathsf{y} + u[i, :]$
    **return** $\mathsf{y}$
```

The above pseudocode uses the following notation conventions:

- Given matrix $z$, the notation $z[i, j]$ denotes accessing the $i$-th row and $j$-th column of $z$.
- $z[i, :]$ denotes the row vector corresponding to the $i$-th row of $z$; $z[:, j]$ is the column vector corresponding to the $j$-th column
- We denote by $\mathsf{CircularBuffer}(n)$ the creation of a new $n$-sized circular buffer. Appending an item to the buffer is accomplished through concatenation operator $\parallel$. After $n$ items have been appended to a buffer, it will overwrite the first record in the buffer, and so on