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

Lorenzo Martinico[1]* , Aydin Abadi[2]**, Thomas Zacharias[1] ***, and Thomas Win[3]†

[1] University of Edinburgh
[2] University College London
[3] University of the West of England

Abstract. 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, researchers have proposed various ad-hoc programs that require being executed on users' smartphones. Nevertheless, the existing solutions have two primary limitations: (1) *lack of generality*: for each type of analytic task, a certain kind of data needs to be sent to an analyst; (2) *lack of transparency*: parties who provide data to an analyst are not necessarily infected individuals; therefore, infected individuals' data can be shared with others (e.g., the analyst) without their fine-grained and direct consent. In this work, we present Glass-Vault, a protocol that addresses both limitations simultaneously. It allows an analyst to run authorised programs over the collected data of infectious users, without learning the input data. Glass-Vault relies on a new variant of generic Functional Encryption that we propose in this work. This new variant, called DD-Steel, offers these two additional properties: dynamic and decentralised. We illustrate the security of both Glass-Vault and DD-Steel in the Universal Composability setting. Glass-Vault is the first UC-secure protocol that allows analysing the data of Exposure Notification users in a privacy-preserving manner. As a sample application, we indicate how it can be used to generate "infection heatmaps".

Keywords: Automated Exposure Notification · Secure Analytics · Functional Encryption · Privacy · Universal Composability.

1 Introduction

The Coronavirus (COVID-19) pandemic has been significantly affecting individuals' personal and professional lives as well as the global economy. The risk

* email: lorenzo.martinico@ed.ac.uk, orcidID: 0000-0002-4968-674X
** email: aydin.abadi@ucl.ac.uk, orcidID: 0000-0002-1414-8351
*** email: thomas.zacharias@ed.ac.uk, 0000-0002-5022-8543
† email: thomas.win@uwe.ac.uk, 0000-0002-4977-0511

of COVID-19 transmission is immensely high among people in close proximity. Tracing those individuals who have been near recently infected people (Contact Tracing) and notifying them of a close contact with an infectious individual (Exposure notification) is one of the vital approaches to efficiently diminishing the spread of COVID-19 [22]. These practices allow 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. Researchers have proposed numerous solutions that can be installed on users' smartphones to improve the efficiency of data collection through automation. Most of the proposed solutions attempt to implement privacy-preserving techniques, such as hiding infected users' contact graphs or adopting designs that prevent tracking of non-infected users [32], to engender sufficient adoption throughout the population [29]. 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 [30].

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 records (e.g., a popluation's average age and common risk factors, or people's previous exposure to infections). Despite the importance of this type of solutions, 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/encoding 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. The solutions proposed in [15, 30] are examples of such ad-hoc analytics protocols. It is desirable that all kinds of user data, independent of their format could be securely collected and transmitted through a unified protocol. In concrete terms, such a unified protocol will (a) provide a generic framework for scientists and health authorities to focus on the analysis of data without having to design ad-hoc security protocols and (b) relieve users from installing multiple applications on their devices to concurrently run data capturing programs, which could 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 by their contact tracing applications. In some of these schemes, the data does not even originate directly from the users, but from a third-party data collector, e.g., a mobile service provider or national health service or even security services [8].

The importance of trustworthy access to raw health data has been recognised by the UK's National Health Service (NHS). It has recently established a large-scale mechanism called "Trusted Research Environments" (TRE) that lets health data be analysed transparently and securely, by authorised researchers [6]. However, even this advanced scheme does not give individuals the possibility to explicitly choose and withdraw consent on their data being used as part of these analytics programs, a right that privacy legislations across the world have increasingly begun to recognise [23].

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. It 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 consensually choose what data to share, and forcing an analyst to execute only those computations authorised by a sufficient number of users, without being able to learn anything about the users' inputs beyond the result.

In this work, we formally define Glass-Vault in the Universal Composability (UC) paradigm and prove its security. We consider a setting where both analysts and users are semi-honest parties who may collude with each other to learn additional information about the data beyond what the analysts are authorised to compute. Furthermore, we allow the adversary to dynamically corrupt users, but assume a static set of corrupted analysts. To have an efficient protocol that can offer all the above features, we construct Glass-Vault by carefully combining pre-existing Exposure Notification algorithms with an extension of the generalised functional encryption proposed by Bhatotia et al. [12]. We extend their construction into a novel protocol called DD-Steel, which allows a functional key to be generated in a distributed fashion while letting any authorised party freely join a key generator committee. We believe DD-Steel is of independent interest. As a concrete application of Glass-Vault, we show how it can be utilised to help the analyst identify clusters of infections through a heatmap.

The rest of the paper has been organised as follows. Section 2 presents an overview of related work and provides key concepts we rely on. Section 3 presents a formal definition of the generic Functional Encryption's new variant, called DD-FESR, along with DD-Steel, a new protocol that realises DD-FESR. Section 4 presents Glass-Vault's formal definition, Glass-Vault protocol, and its security proof. Section 5 provides a concrete example of a computation that an analyst in Glass-Vault can perform, i.e., infection heatmaps. Appendices A and B provide more detail on the underlying key concepts, while Appendix C elaborates on how the infection heatmaps can be implemented. Appendix D provides more detail about the concept used in DD-FESR and DD-Steel.

2 Background

We now give a brief overview of existing literature regarding our security framework (2.1) and the building blocks for our constructions (2.2-2.4)

2.1 Universal Composability

There are various paradigms via which a cryptographic protocol might be defined and proven, such as game-based or simulation-based paradigms. Universal Composability (UC), introduced by Canetti [16], is a simulation-based model that ensures security even if multiple instances of a protocol run in parallel. 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 a 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, via 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. [11] 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

In this section, we present an overview of various variants of privacy-preserving solutions for contact tracing.

Centralized vs Decentralized Contact Tracing. Within the first few months of the COVID-19 pandemic, a large number of theoretical (e.g., [10, 20, 37, 42]) and practical (e.g., [1, 4, 5, 7, 27]) automated contact tracing solutions were quickly developed by governments, industry, and academic communities. Most designs 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 by a trusted health authority and distributed to contact tracing users. In 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 an 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. In this setting, the authority is able to construct users' contact graphs, which allows it to further analyse users' movements and interactions, at the cost of privacy. On the other hand, the decentralised users download the list of broadcasts from exposed users and compare it with their own local lists. This guarantees additional privacy compared to centralised systems (although several attacks are still possible, see [32]), while also preventing the health authority from running large scale analysis on population infection which would be possible in a centralised system. Despite this, the adoption of decentralised systems such as DP-3T [43] has been more widespread due to technical restrictions and political decisions forced by smartphone manufacturers [9]. 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 [10, 44, 45].

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. While the above work is unique in formalising Exposure notification, a UC formalisation of the related problem of proximity testing has been given in [40], based on the reduction to Private Equality Testing by Narayanan et al. [35].

Automated Data Analysis. A few attempts have been made to develop automated systems which can analyse population behaviour to better understand the spread of the virus. The solution proposed in Bruni et al. [15] displays the development of virus hotspots, as a heatmap. In this system, there are two main players; namely, the health authority and a mobile phone provider, each of which has a set of data that they have independently collected from their users. Their goal is to find (only) the heatmap in a privacy-preserving manner, i.e., without revealing their input in plaintext to their counterparty. To achieve its goal, the system uses (a computationally expensive) homomorphic encryption, differential privacy, and the matrix representation of inputs. Thus, in this system, (i) the two parties run the computation on users' data, without having their fine-grained consent and (ii) each party's input has to be encoded in a specific way, i.e., it must be encrypted and represented as a matrix.

The protocols in [13, 28, 30] allow users to provide their encoded data to a server for a specific analysis. Specifically, Lueks et al. [30] design a privacy-preserving "Presence-Tracing" system which notifies people who were in the same venue as an infected individual. The proposed solution mainly uses identity-based encryption, hash function, and authenticated encryption to achieve its goal and encode users' input data. Biasse et al. [13] design a privacy-preserving scheme to anonymously collect information about a social graph of users. In this solution, a central server can construct an anonymous graph of interactions between users which would let the server understand the progression of the virus among users. This solution is based on zero-knowledge proofs, digital signatures, and RSA-based accumulators. Günther et al. [28] propose a privacy-preserving scheme between multiple non-colluding servers to help epidemiologists simulate and predict future developments of the virus. This scheme relies on heavy machineries such as oblivious shuffling, anonymous credentials, and generic multi-party computation.

In all of the above solutions, the parties need to encode their inputs in a certain way to support the specific computation that is executed on their inputs, and thus do not support generality. Additionally, not all systems allow the users to opt-out of the computation, and are therefore not transparent.

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. [36] in the GUC model. This definition provides a basic abstraction called the *global attestation functionality* (G_{att}, described in Supporting material B.1), that casts the core services a wide class of real-world attested execution processors offer. G_{att} has been used by various protocols both in the GUC model (e.g., in [41, 46]) and recently in the UCGS model (in [12]). TEEs have been used in a few protocols to implement Contact Tracing solutions, e.g., in [21, 31, 39].

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 Supporting material A.

An FE scheme can also be thought of as a way to implement access control to an ideal repository [34]. Since the introduction of FE, various variants of FE

schemes have been proposed, such as those 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, 33] 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 $\mathtt{F} = \{\mathrm{F} \mid \mathrm{F} : \mathcal{X} \times \mathcal{S} \times \mathcal{R} \to \mathcal{Y} \times \mathcal{S}\}$, 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_{att}). By introducing functions with state and randomness, FESR allows computing a broader class of functions than most other FE schemes. Moreover, other appealing properties of the scheme include its secure realisation in the UC paradigm, and that it can be easily extended to support multiple inputs (as we show in Subsection 3.4).

3 Dynamic and Decentralised FESR (*DD*-FESR) and Steel (*DD*-Steel)

In this section, we present the ideal functionality *DD*-FESR and the protocol that realises it, *DD*-Steel. As we stated earlier, they are built upon the original functionality FESR and protocol Steel, respectively. In the formal descriptions, we will highlight the main changes that we have applied to the original scheme (in [12]) in yellow. For conciseness, we omit any reference to UC-specific machinery such as "session ids" unless it is required to understand the specifics of our protocol.

3.1 The ideal functionality *DD*-FESR

In this subsection, we extend FESR into a new functionality *DD*-FESR to capture two additional properties from the functional encryption literature:

– *Decentralisation* (introduced in [24]) allows a set of encryptors to be in control of functional 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 $\mathsf{A} \in \mathbf{A}$ 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 [25]) 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 preexisting **A** members, or with

an empty set that is gradually filled through Setup calls. Our current scheme is permissionless, meaning anyone can register as a new party.

Functionality DD-FESR$[\mathtt{F}, \mathbf{A}, \mathbf{B}, \mathsf{C}]$

The functionality is parameterised by the randomized function class $\mathtt{F} = \{\mathrm{F} \mid \mathrm{F} : \mathcal{X} \times \mathcal{S} \times \mathcal{R} \to \mathcal{Y} \times \mathcal{S}\}$, over state space $\mathcal{S}$ and randomness space $\mathcal{R}$, and by three distinct types of party entities $\mathsf{A} \in \mathbf{A}, \mathsf{B} \in \mathbf{B}, \mathsf{C}$ 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
$\mathtt{F}^+$	Union of the allowable functions and leakage function, i.e., $\mathtt{F} \cup \mathrm{F}_0$
$\mathsf{setup}[\cdot] \leftarrow \mathsf{false}$	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 authorised functions' states for all decryption parties
$\mathcal{KS}[\cdot] \leftarrow []$	Table of key share generator for each (decryptor, function) pair

```
On message SETUP from $P$:
  assert $\mathsf{setup}[P] = \mathsf{false}$
  send $(\textsc{setup}, P)$ to $\mathcal{A}$ and await OK; then
  if $P.\mathsf{code} = \mathsf{A}$ then $\mathbf{A} \leftarrow \mathbf{A} \parallel P$
      // Party membership is determined by the code in the UC identity tape
  else if $P.\mathsf{code} = \mathsf{B}$ then $\mathbf{B} \leftarrow \mathbf{B} \parallel P$
  else return
  $\mathsf{setup}[P] \leftarrow \mathsf{true}$
On message $(\textsc{KeyShareGen}, \mathrm{F}, \mathsf{B})$ from $\mathsf{A} \in \mathbf{A}$:
  if $\mathsf{A} \in \hat{\mathbf{A}}$ then
      // The adversary can only block key generation for corrupted parties
      send $(\textsc{keysharequery}, \mathrm{F}, \mathsf{A}, \mathsf{B})$ to $\mathcal{A}$ and await OK; then
  if $(\mathrm{F} \in \mathtt{F}^+ \wedge \mathsf{setup}[\mathsf{A}] \wedge \mathsf{setup}[\mathsf{B}])$ then
      $\mathcal{KS}[\mathsf{B}, \mathrm{F}] \leftarrow \mathcal{KS}[\mathsf{B}, \mathrm{F}] \parallel \mathsf{A}$
      // We store the identity of all parties in $\mathbf{A}$ who authorised F for B
      send $(\textsc{keysharegen}, \mathrm{F}, \mathsf{A}, \mathsf{B})$ to $\mathsf{B}$
      if $\mathsf{B} \in \hat{\mathbf{B}} \vee \mathsf{A} \in \hat{\mathbf{A}}$ then send $(\textsc{keysharegen}, \mathrm{F}, \mathsf{A}, \mathsf{B})$ to $\mathcal{A}$
      else send $(\textsc{keysharegen}, \perp, \mathsf{A}, \mathsf{B})$ to $\mathcal{A}$
On message $(\textsc{Encrypt}, \mathrm{x}, k)$ from party $P \in \{\mathbf{A} \cup \mathbf{B}\}$:
  if $(\mathsf{setup}[P] \wedge \mathrm{x} \in \mathcal{X} \wedge k$ is an integer$)$ then
      compute $\mathsf{h} \leftarrow \mathtt{getHandle}$
      // Generate a unique index, h, by running the subroutine getHandle
```

```
        $\mathcal{M}[\mathsf{h}] \leftarrow (\mathrm{x}, k)$
        **send** $(\text{ENCRYPTED}, \mathsf{h})$ **to** $P$
    **else**
        **send** $(\text{ENCRYPTED}, \perp)$ **to** $P$
*On message* $(\text{DECRYPT}, \mathrm{F}, \mathsf{h})$ *from party* $\mathsf{B} \in \mathbf{B}$*:*
    $(\mathrm{x}, k) \leftarrow \mathcal{M}[\mathsf{h}]; \mathrm{y} \leftarrow \perp;$
    **if** $\mathrm{F} = \mathrm{F}_0$ **then**
        $\mathrm{y} \leftarrow |\mathrm{x}|$
    **else if** $((|\mathcal{KS}[\mathsf{B}, \mathrm{F}]| \geq k \wedge \forall \mathsf{A} \in \mathcal{KS}[\mathsf{B}, \mathrm{F}] : \mathsf{setup}[\mathsf{A}] \wedge \forall \mathsf{A}$ are distinct $)\vee$
        $\vee(\mathsf{B} \in \hat{\mathbf{B}} \wedge |\hat{\mathbf{A}}| \geq k))$ **then**
        // There are at least $k$ functional key shares, all generated by correctly setup
    parties, OR B and at least $k$ parties in $\mathbf{A}$ are corrupted
        $\mathsf{s} \leftarrow \mathcal{P}[\mathsf{B}, \mathrm{F}]$
        $\mathrm{r} \xleftarrow{\$} \mathcal{R}$
        $(\mathrm{y}, \mathsf{s}') \leftarrow \mathrm{F}(\mathrm{x}, \mathsf{s}; \mathrm{r})$
        $\mathcal{P}[\mathsf{B}, \mathrm{F}] \leftarrow \mathsf{s}'$
    **return** $(\text{DECRYPTED}, \mathrm{y})$
*On message* $(\text{CORRUPT}, P)$ *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 $\mathbf{A}$ to ensure
    correctness
        **return** $\{(\mathsf{B}, \mathrm{F}) | P \in \mathcal{KS}[\mathsf{B}, \mathrm{F}]\}$
    **if** $P \in \mathbf{B}$ **then**
        $\hat{\mathbf{B}} \leftarrow \hat{\mathbf{B}} \parallel P$
        **return** $\mathcal{KS}[P, \cdot]$
```

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 $\mathbf{A}$, and the master secret key is securely stored in an enclave running program $\mathsf{prog}_{\mathsf{KME}}$ 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 $\mathsf{prog}_{\mathsf{FE}[\mathrm{F}]}$ enclave, which can then decrypt any A's encrypted inputs x and will ensure that only value y of $(\mathrm{y}, \mathsf{s}') \leftarrow \mathrm{F}(\mathrm{x}, \mathsf{s}; \mathrm{r})$ 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 as described, except for a few crucial differences. Party C, who is now untrusted, still runs the public key encryption parameter generation within an enclave and distributes it to a party A or B when they first join the protocol. Each party A also generates a digital signature key pair locally and includes a key policy k along with their ciphertext. Note that for simplicity our current version uses an integer k to associate with each message, 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 $\mathsf{KeyShareGen}(\mathrm{F}, \mathsf{B})$ 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 $\mathbf{A}$; if all checks are passed, then it will distribute the master secret key to $\mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\mathrm{F}]}$, along with the length of recorded key shares k_{F}. $\mathsf{prog}_{\mathsf{FE}^{\mathsf{VK}}[\mathrm{F}]}$ 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 $\mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}$ enclave remains as in FESR.

The protocol DD-Steel makes use of the global attestation functionality G_{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 Supporting material 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-$\mathsf{Steel}[\mathtt{F}, \mathsf{PKE}, \Sigma, \mathsf{N}, \lambda]$

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

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, $P$) from $\mathcal{SC}^P$:
  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 $\mathsf{VK}$
    $\mathsf{eid}_{\mathsf{KME}} \leftarrow G_{\mathsf{att}}.\mathsf{install}(\mathsf{C.sid}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}})$
    $(\mathsf{mpk}, \sigma_{\mathsf{KME}}) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{KME}}, (\mathrm{init}, \mathsf{crs}, \mathsf{C.sid}))$
  if $P.\mathsf{code} = \mathsf{A}$ then
    send $(\textsc{Setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ to $\mathcal{SC}_P$
  else if $P.\mathsf{code} = \mathsf{B}$ then
    send $(\textsc{Setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ to $\mathcal{SC}_P$ and receive $(\textsc{provision}, \sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD})$
    $(\mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{sk}}) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{KME}}, (\mathrm{provision}, (\sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}})))$
```

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

```
$\underline{\mathsf{prog}_{\mathsf{KME}^{\mathsf{VK}}}}$
on input init
    $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.PGen}(1^\lambda)$
    $\textbf{return}\ \mathsf{pk}$
on input $(\text{provision}, (\sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}))$:
    $\mathsf{vk}_{\mathsf{att}} \leftarrow G_{\mathsf{att}}.\mathsf{vk}_{\mathsf{att}}$; $\textbf{fetch}\ \mathsf{crs}, \mathsf{idx}, \mathsf{sk}$
    $\textbf{assert}\ \Sigma.\mathsf{Vrfy}(\mathsf{vk}_{\mathsf{att}}, (\mathsf{idx}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}, (\mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}), \sigma_{\mathsf{DE}})$
    $\mathsf{ct}_{\mathsf{key}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{KD}, \mathsf{sk})$
    $\textbf{return}\ \mathsf{ct}_{\mathsf{key}}$

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

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

3.3 Proof of security

We now formally state the security guarantees of DD-Steel as a Theorem:

Theorem 1. *For a class of functions* $\mathtt{F}$*, CCA-secure encryption scheme* PKE*, EU-CMA secure signature scheme* Σ*, and non-interactive zero-knowledge proof system* N*, Protocol* DD-$\mathsf{Steel}[\mathtt{F}, \mathsf{PKE}, \Sigma, \mathsf{N}, \lambda]$ *UC-realises ideal functionality* DD-$\mathrm{FESR}[\mathtt{F}, \mathbf{A}, \mathbf{B}, \mathbf{C}]$*, in the presence of global functionality* G_{att}*.*

Proof. We first construct a simulator, $\mathcal{S}_{DD\text{-FESR}}$. For simplicity of exposition, we use the simulator $\mathcal{S}_{\mathrm{FESR}}$ in [12] as a subroutine to $\mathcal{S}_{DD\text{-FESR}}$. We instantiate $\mathcal{S}_{\mathrm{FESR}}$ such that shared functionalities (e.g., the secure channels between multiple parties) are implemented by $\mathcal{S}_{DD\text{-FESR}}$, so that it can intercept messages to the parties whose behaviour is simulating and act accordingly. $\mathcal{S}_{DD\text{-FESR}}$ also acts as the ideal functionality in the eyes of the $\mathcal{S}_{\mathrm{FESR}}$ simulator. We reproduce the original $\mathcal{S}_{\mathrm{FESR}}$, with appropriate modifications to conform to the message syntax of DD-FESR and DD-Steel, in Supporting material D.

We assume that at least one party in the set $\mathbf{B} \cup \mathsf{C}$ is corrupted at the start of the protocol. We choose this party, GG, to install all G_{att} enclaves for all participants in the protocol, be they honest or corrupted. Due to the property of anonymous attestation guaranteed by G_{att}, the simulator can install all programs on the same machine to produce the attested trace of the real-world protocol, as long as it does not allow GG to execute other parties' enclaves on its own initiative (we use the table $\mathcal{G}$ from simulator $\mathcal{S}_{\mathrm{FESR}}$ to keep track of which party installed each enclave). Similar to the original simulator, we use the shorthand $\mathsf{output} \leftarrow G_{\mathsf{att}}.\mathsf{command}(\mathsf{input})$ to indicate "**simulate sending** $(\textsc{command}, \mathsf{input})$ **to** G_{att} **through** GG and **receive** output"; note, in [12], the message was always sent from B instead, given the simpler setting of one honest C and one corrupted B in their proof.

We also give $\mathcal{S}_{DD\text{-FESR}}$ white-box access to $\mathcal{S}_{\mathrm{FESR}}$, letting the former freely access the internal tapes of the latter. We mark the names of the variables that are read from $\mathcal{S}_{\mathrm{FESR}}$ in the state variable declaration below. In particular, we use the internal values of $\mathcal{S}_{\mathrm{FESR}}$ to keep track of messages sent to the enclaves and their attestation signatures. For all calls to an enclave, the $\mathcal{S}_{DD\text{-FESR}}$ simulator always activates $\mathcal{S}_{\mathrm{FESR}}$ so that these internal variables can be updated.

Simulator $\mathcal{S}_{DD\text{-FESR}}$

State variables	Description
$K \leftarrow \{\}$	set of **A** keypairs and $\mathcal{F}_{\mathsf{CERT}}$ certificates
$KS \leftarrow \{\}$	set of generated keyshares
$\mathsf{GG} \leftarrow \perp$	The corrupted party on which we run simulated enclaves
$\mathcal{G} = \mathcal{S}_{\text{FESR}}.\mathcal{G}$	Collects all messages sent to G_{att} and its response
$\mathcal{B} = \mathcal{S}_{\text{FESR}}.\mathcal{B}$	Collects all messages signed by G_{att}
$\mathsf{crs} = \mathcal{S}_{\text{FESR}}.\mathsf{crs}$	Simulated common reference string

```
On message $(\textsc{Setup}, P)$ from $DD$-FESR:
  if $P.\mathsf{code} = \mathsf{A}$ then
      if $P$ is honest then
          if $\mathsf{C}$ is honest then
              send $(\textsc{setup}, P)$ to $\mathcal{S}_{\text{FESR}}$
          else
              send $(\textsc{Setup}, \mathsf{A.pid})$ to $\mathcal{SC}^{\mathsf{C}}$ and receive $\mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}}$
              assert $(\mathsf{C.sid}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{vk}}}, \mathsf{mpk}) \in \mathcal{B}[\sigma_{\mathsf{KME}}]$
          $\mathsf{vk}_{\mathsf{att}} \leftarrow G_{\mathsf{att}}.\mathsf{getPK}()$
          send $\textsc{GetK}$ to $\mathcal{F}_{\mathsf{CERT}}$ and receive $\mathsf{VK}$
          $(\mathsf{vk}_{\Sigma}, \mathsf{sk}_{\Sigma}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$
          simulate sending $(\textsc{Sign}, \mathsf{vk}_{\Sigma})$ to $\mathcal{F}_{\mathsf{CERT}}$ through P and receive cert
          $K[P] \leftarrow (\mathsf{vk}_{\Sigma}, \mathsf{sk}_{\Sigma}, \mathsf{cert})$
          send OK to $DD$-FESR
      else
          if $\mathsf{C}$ is honest then
              simulate sending $(\textsc{setup}, P)$ to $P$ on behalf of $\mathcal{Z}$
              await for message $(\textsc{setup}, P)$ on $\mathcal{SC}^{\mathsf{C}}_{P}$
              send $(\textsc{setup}, P)$ to $\mathcal{S}_{\text{FESR}}$
              send OK to $DD$-FESR
          else
              simulate sending $(\textsc{setup}, P)$ to $P$ on behalf of $\mathcal{Z}$
              await for message $(\textsc{Sign}, \mathsf{vk}_{\Sigma})$ from $P$ to $\mathcal{F}_{\mathsf{CERT}}$
              send OK to $DD$-FESR
  else if $P.\mathsf{code} = \mathsf{B}$ then
      if $P$ is honest then
          if $\mathsf{C}$ is honest then
              notify $\mathcal{S}_{\text{FESR}}$ that $(\textsc{install}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{vk}}})$ was sent from $P$ to $G_{\mathsf{att}}$ and
  capture response $\mathsf{eid}_{\mathsf{DE}}$
              send $(\textsc{setup}, P)$ to $\mathcal{S}_{\text{FESR}}$ and receive $\mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}}$
              notify $\mathcal{S}_{\text{FESR}}$ that $(\textsc{resume}, \text{init-setup}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}, P.\mathsf{sid})$ was sent from
  $P$ to $G_{\mathsf{att}}$ and capture response $(\mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}), \sigma_{\mathsf{init}}$
              send $(\textsc{provision}, \sigma_{\mathsf{init}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD})$ to $\mathcal{S}_{\text{FESR}}$ and receive $(\textsc{provision}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}})$

              notify $\mathcal{S}_{\text{FESR}}$ that $(\textsc{resume}, \text{complete-setup}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}})$ was sent from
  $P$ to $G_{\mathsf{att}}$
              send OK to $DD$-FESR
          else
              simulate sending $(\textsc{setup}, P)$ to $P$ on behalf of $\mathcal{Z}$
```

```
            await for SETUP message on $\mathcal{SC}_P^{\mathsf{C}}$
            send $(\textsc{setup}, P)$ to $\mathcal{S}_{\text{FESR}}$
    else
        if $\mathsf{C}$ is honest then
            send $(\textsc{setup}, P)$ to $\mathcal{S}_{\text{FESR}}$
            send OK to $DD$-FESR
        else
            simulate sending $(\textsc{setup}, P)$ to $P$ on behalf of $\mathcal{Z}$
            await for $(\textsc{resume}, \text{complete-setup}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}})$ from $P$ to $G_{\mathsf{att}}$
            $(\mathsf{idx}, \mathsf{eid}, \mathsf{prog}, \mathsf{ct}_{\mathsf{key}}) \leftarrow \mathcal{B}[\sigma_{\mathsf{KME}}]$
            assert $\mathsf{idx} = P.\mathsf{sid} \land \mathsf{prog} = \mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}} \land (\sigma_{\mathsf{KME}}, \cdot, \mathsf{ct}_{\mathsf{key}}) \in \mathcal{G}[\mathsf{eid}].\mathsf{resume}$
            send OK to $DD$-FESR
On message $(\textsc{Sign}, \mathsf{vk}_\Sigma)$ from corrupted party $P$ to $\mathcal{F}_{\mathsf{CERT}}$:
  forward $(\textsc{Sign}, \mathsf{vk}_\Sigma)$ and receive response $\mathsf{cert}$
  $K[P] \leftarrow (\mathsf{vk}_\Sigma, \perp, \mathsf{cert})$
On message $(\textsc{keysharequery}, x, \mathsf{A}, \mathsf{B})$ from $DD$-FESR:
  // A is corrupted
  send $(\textsc{KeyShareGen}, x, \mathsf{A}, \mathsf{B})$ to $\mathsf{A}$ on behalf of $\mathcal{Z}$ and await for $(\textsc{KeyShareGen}, x, \mathsf{A}, \mathsf{B})$
  from $\mathsf{A}$ to $\mathcal{SC}^{\mathsf{B}}$
  return OK
On message $(\textsc{keysharegen}, f, \mathsf{A}, \mathsf{B})$ from $DD$-FESR:
  if $\mathsf{A}$ is honest then
      if $\mathsf{B}$ is honest then $\mathrm{F} \leftarrow \mathrm{F}_0$
      else $\mathrm{F} \leftarrow f$
      $(\mathsf{vk}_\Sigma, \mathsf{sk}_\Sigma, \mathsf{cert}) \leftarrow K[\mathsf{A}]$
      $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_\Sigma, \mathrm{F}, \mathsf{B})$
      $KS[\mathrm{F}, \mathsf{A}, \mathsf{B}] \leftarrow \sigma$
      send $(\textsc{KeyShareGen}, \mathrm{F}, \sigma, \mathsf{vk}_\Sigma, \mathsf{cert})$ to $\mathcal{SC}_{\mathsf{A}}^{\mathsf{B}}$
On message $(\textsc{resume}, \mathsf{eid}, \mathsf{input})$ from corrupted party $P$ to $G_{\mathsf{att}}$:
  if $\mathcal{G}[\mathsf{eid}].\mathsf{install}[1] = \mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}} \land \mathsf{input}[0] = \text{provision}$ then
      $(\text{provision}, \mathcal{KS}_{\mathrm{F}}, \mathsf{eid}_{\mathrm{F}}, \mathsf{pk}_{\mathsf{FD}}, \sigma_{\mathrm{F}}, \mathrm{F}, \mathsf{pid}) \leftarrow \mathsf{input}$
      for $(\sigma, \mathsf{vk}_\Sigma, \mathsf{cert}) \in \mathcal{KS}_{\mathrm{F}}$ do
          assert $(\mathsf{vk}_\Sigma, \cdot, \mathsf{cert}) = K[\mathrm{F}, \mathsf{A}, P]$ for some $\mathsf{A}$
          if $\sigma \notin KS[\mathrm{F}, \mathsf{A}, P]$ then
              $KS[\mathrm{F}, \mathsf{A}, P] \leftarrow \sigma$
              send $(\textsc{KeyShareGen}, \mathrm{F}, \sigma, \mathsf{vk}_\Sigma, \mathsf{cert})$ to $DD$-FESR through $\mathsf{A}$ and
  await $(\textsc{KeyShareQuery}, \mathrm{F}, \mathsf{A}, P)$; then send OK to $DD$-FESR
  send $(\textsc{resume}, \mathsf{eid}, inp)$ to $\mathcal{S}_{\text{FESR}}$
On message $(\textsc{encrypt}, \mathsf{input})$ from $\mathcal{S}_{\text{FESR}}$ on behalf of $P$:
  send $(\textsc{encrypt}, \mathsf{input})$ to $DD$-FESR through $P$ and receive $(\textsc{encrypted}, \mathsf{output})$
  send $(\textsc{encrypted}, \mathsf{output})$ to $\mathcal{S}_{\text{FESR}}$ on behalf of $DD$-FESR
On message $(\textsc{decrypt}, \mathsf{input})$ from $\mathcal{S}_{\text{FESR}}$ on behalf of $P$:
  send $(\textsc{decrypt}, \mathsf{input})$ to $DD$-FESR through $P$ and receive $(\textsc{decrypted}, \mathsf{output})$
  send $(\textsc{decrypted}, \mathsf{output})$ to $\mathcal{S}_{\text{FESR}}$ on behalf of $DD$-FESR
On message *:
  forward * to $\mathcal{S}_{\text{FESR}}$
```

We now show, via a series of hybrid experiments, that given the above simulator, the real and ideal worlds are indistinguishable from the environment's viewpoint. We begin with the real-world protocol, which can be considered as *Hybrid* 0.

Hybrid 1 consists of the ideal protocol for DD-FESR, which includes the relevant dummy parties, and the simulator $\mathcal{S}'_{DD\text{-FESR}}$, which on any message from the environment ignores the output of the ideal functionality, and faithfully reproduces protocol DD-Steel. The equivalence between *Hybrids* 0 and 1 is trivial due to the behaviour of $\mathcal{S}'_{DD\text{-FESR}}$.

Hybrid 2 replaces all operations of $\mathcal{S}'_{DD\text{-FESR}}$ where the protocol DD-Steel behaves in the same way as Steel (except that it sends messages with the full set of arguments expected by DD-Steel rather than those in Steel, and receives the equivalent DD-Steel return values) with a call to an emulated $\mathcal{S}_{\text{FESR}}$ as defined in Supporting Material D. Due to the security proof of the Steel protocol in [12], we now use the simulator of *Hybrid* 2 to simulate FESR with respect to protocol Steel, making the two hybrids indistinguishable. An environment that is able to distinguish between the two hybrids could create an adversary that can distinguish between executions of FESR and Steel; but due to the UC emulation statement, no such environment can exist in the presence of $\mathcal{S}_{\text{FESR}}$. The reduction to $\mathcal{S}_{\text{FESR}}$ greatly simplifies the current proof, as we are guaranteed the security of the secure key provisioning and decryption due to the similarities of these two phases of the protocols between Steel and DD-Steel.

Hybrid 3 modifies the simulator of *Hybrid* 2 by replacing all the signature verification operations for attestation signatures in DD-FESR with a table lookup from $\mathcal{S}_{\text{FESR}}.\mathcal{B}$. The new table lookups for attestation signatures complement the ones enacted by $\mathcal{S}_{\text{FESR}}$, while capturing behaviour that is unique to DD-Steel. Similar to [12, Lemma 2], if the environment can distinguish between this hybrid and the previous one, it can construct an adversary to break the unforgeability of signatures.

Hybrid 4 modifies the simulator of *Hybrid* 3 by replacing KeyShareGen and KeyShareQuery requests for any functions with a request for a dummy function (such as the natural leakage function F_0), for all these requests where both the encryptor and the decryptor are honest. The environment is not able to distinguish between the two hybrids due to the security of the secure channel functionality (as defined in Supporting Material B.4): the secure channel only leaks the length of a message exchanged between sender and receiver, and assuming that we represent functions with a fixed-length string (such as a hash of it's code), the leakage between this hybrid and the previous one is indistinguishable.

Hybrid 5 adds an additional check to the simulator of *Hybrid* 4 before it can run the provision command on enclave $\mathsf{prog}_{\mathsf{DE}^{\mathsf{VK}}}$ through the internal $\mathcal{S}_{\text{FESR}}$ simulator. The check ensures that all keyshares passed by the malicious decryptor to the enclave are signed by a party who has first registered their verification key with the certificate authority. Then, if the signature has not been generated through a call to the ideal functionality but rather through a local signing operation, the simulator notifies the ideal functionality to update its internal keyshare count. This hybrid essentially replaces the algorithmic signature verification operations in the previous one with two table lookups (for both verification key certification and keyshare authenticity). If an adversary was able to bypass the checks by providing either a certificate that wasn't produced by the ideal functionality or a keyshare that didn't match with the triple of $(\mathrm{F}, \mathsf{A}, \mathsf{B})$, they would be able to create an adversary that could break the unforgeability of signature scheme Σ in the same manner as in Hybrid 3. Thus the hybrid is indistinguishable from the previous one.

The simulator defined in *Hybrid* 5 is identical to $\mathcal{S}_{DD\text{-FESR}}$; thus, it holds that DD-Steel UC-emulates DD-FESR. $\square$

3.4 Turning Stateful functions into Multi-Input

As pointed out in [12], FESR subsumes Multi-Input Functional Encryption [26] 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 $\mathrm{F} : (\underbrace{\mathcal{X} \times \cdots \times \mathcal{X}}_{n}) \rightarrow \mathcal{Y}$ we define the following stateful functionality:

```
function $\mathsf{Agg}_{\mathrm{F},n}(\mathrm{x}, \mathsf{s})$
    if $|\mathsf{s}| < n$ then return $(\bot, \mathsf{s} \parallel \mathrm{x})$
    else return $(\mathrm{F}(\mathsf{s}||\mathrm{x}), \emptyset)$
        // 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. There are several possible extensions to the above compiler:

1. In $\mathsf{Agg}_{\mathrm{F},n}(\cdot)$, 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.
2. The compiler can be easily extended to multi-input *stateful* functionalities, by keeping a list of inputs (as a field) within the state array and not discarding the state on the n-th invocation of the compiler.
3. 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 $[\![\mathcal{X}]\!] \to \mathcal{Y}$). On the first (integer) input to the aggregator, we set it as a special field n in the state, and for the next $n-1$ calls we simply append the inputs to the state, while returning the empty value. On the nth call, we execute the function on the stored state field (now containing n entries), erase the state and index from memory and wait for the next call to set a new value to n.

In the next section, we will use the compiler $\mathsf{AggS}_{\mathrm{F}}$, which combines the above defined compiler extensions 2 and 3 to compute any stateful function F with variable number of inputs ($\mathrm{F} : [\![\mathcal{X}]\!] \times \mathcal{S} \times \mathcal{R} \to \mathcal{Y} \times \mathcal{S}$) from DD-FESR.

4 The Glass-Vault platform

In this section, first, we provide the formal definition of "Analysis-augmented Exposure Notification" (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 EN^+, i.e., $\mathcal{F}_{\mathrm{EN}^+}$.

4.1 Analysis-augmented Exposure Notification (EN^+)

Since EN^+ is built upon EN, we first re-state relevant notions used in the UC modelling of EN. Specifically, EN relies on the time functionality $\mathbb{T}$ and the "physical reality" functionality $\mathbb{R}$ that we present in Supporting Material 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. The functionality only accepts new events if they are "physically sensible", and can send the entire list of events to some privileged entities such as ideal functionalities. Using $\mathbb{T}$ and $\mathbb{R}$ as subroutines, the functionality of EN, i.e., $\mathcal{F}_{\mathrm{EN}}$ (presented formally in Supporting Material B.8), is defined in terms of a risk estimation function ρ, a leakage function $\mathcal{L}$, a set of allowable measurement error functions E, and a set of allowable faking functions Φ. 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 (where an infectious user appears to be within transmission distance from a non-infectious malicious user). The functionality captures a variety of contact tracing protocols and attacker models via 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 EN$^+$ involves an additional entity to the above scheme, namely, analyst $\ddot{\mathsf{A}}$, who wants to learn a certain function, α, on data contributed by exposed users, some of which might be sensitive. Thus, exposed users are 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 now present a formal definition of the ideal functionality for EN$^+$. Highlighted sections of the functionality represent where EN$^+$ diverges from EN.

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

The functionality is parametrised by exposure risk function ρ, a set of allowable error functions E for the physical reality record, a set of faking functions Φ for the adversary to misrepresent the physical reality, and a leakage function $\mathcal{L}$, 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
$\mathbf{SE}$	List of users who have shared their exposure status and time of upload
$\overline{\mathbf{U}}$	List of active users; $\mathbf{SE} \cap \overline{\mathbf{U}} = \emptyset$
$\widetilde{\mathbf{U}}$	List of corrupted users
$\overline{A}$	For each pair of analyst and allowed function, the dictionary $\overline{A}$ contains the users that have authorised this pair
$\widetilde{\ddot{\mathsf{A}}}$	Static set of corrupted analysts
$\mathcal{ST}$	State table for function, analyst pairs

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

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

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 an analyst requests to execute some specific computations on exposed users' data, the users are informed via public announcements. At this stage, users can provide permission tokens to the analyst (in the form of functional keysahres), 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 scheme, the Glass-Vault analyst 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 EN and DD-FESR functionality, the Glass-Vault protocol makes use of the physical reality functionality $\mathbb{R}$, the exposure notification functionality $\mathcal{F}_{\text{EN}}$, and the trusted bulletin board functionality $\mathcal{F}_{\mathsf{TBB}}$, described in Supporting Material 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 class of parameters as defined in $\mathcal{F}_{\text{EN}^+}$, 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 $\ddot{\mathsf{A}}$ to refer to an analyst (corresponding to DD-FESR's decryptor, B). Among other ideal setups, Glass-Vault leverages the exposure notification ideal functionality 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 $\mathcal{F}_{\text{EN}}$
On message ShareExposure from a party P:
  send (ShareExposure, $U$) to $\mathcal{F}_{\text{EN}}$ 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, $\mathsf{h}$)
        erase $u^e_{\mathsf{SEC}}$ and send (Add, $\mathsf{h}$) to $\mathcal{F}_{\mathsf{TBB}}$
  On message ExposureCheck from a party $P$:
    send (ExposureCheck, $U$) to $\mathcal{F}_{\mathrm{EN}}$ and receive $\rho_U$
    if $\rho_U \neq$ error then return $\rho_U$
  On message (RegisterAnalystAccept, $\alpha$, $\ddot{\mathsf{A}}$) from a party $P$:
    send (KeyShareGen, $\mathsf{AggS}_\alpha$, $\ddot{\mathsf{A}}$) to $DD$-FESR
    send (RegisterAnalystAccept, $U$, $\alpha$) to $\ddot{\mathsf{A}}$
  Analyst $\ddot{\mathsf{A}}$:
  On message (RegisterAnalyst, $\alpha$, $\ddot{\mathsf{A}}$) from a party $P$:
    send Setup to $DD$-FESR
    for all exposed $U$ do send (RegisterAnalystRequest, $\alpha$, $\ddot{\mathsf{A}}$, $U$) to $P$
  On message (Analyse, $\alpha$) from a party $P$:
    send Retrieve to $\mathcal{F}_{\mathsf{TBB}}$ and receive $\mathcal{C}$
    send (Encrypt, $(|\mathcal{C}|, 0)$) to $DD$-FESR and receive $\mathsf{h}_n$
    send (Decrypt, $\mathsf{h}_n$, $\mathsf{AggS}_\alpha$) to $DD$-FESR and receive (decrypted, $|\mathcal{C}|$)
    for $\mathsf{h} \in \mathcal{C}$ do
        send (Decrypt, $\mathsf{h}$, $\mathsf{AggS}_\alpha$) to $DD$-FESR and receive (decrypted, y)
    if $\mathrm{y} \neq \perp$ then return (Decrypted, $\alpha$, $P$, y)
```

4.3 Proof of security

We now show that Glass-Vault is secure:

Theorem 2. *Let $\rho, E, \Phi, \Phi^+, \mathcal{L}, \mathcal{L}^+, AF, K,$ and C be parameters such that the following conditions hold: (1) $\Phi \subset \Phi^+$, (2) for every input x, it holds that[4] $\mathcal{L}(x) = \mathcal{L}(\mathcal{L}^+(x))$, 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 set of users U, (3.1) $\phi(\widetilde{\mathrm{R}}_\epsilon)$ does not tamper with sensitive data record $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{SEC}]$ but $\phi^+(\widetilde{\mathrm{R}}_\epsilon)$ does, and (3.2) $\mathcal{L}(x)$ does not contain any instruction to leak the contents of $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{SEC}]$ but $\mathcal{L}^+(x)$ does. Then, it holds that*

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

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

Proof. We construct a simulator $\mathcal{S}_{\mathsf{GV}}$ to prove Theorem 2. The high-level task of our simulator is to *synchronise* the inputs of the analysis 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). The simulator updates a simulated trusted bulletin board by obtaining, through the leakage function, the secret data for all honest users who have shared their exposure, and encrypting it through DD-FESR.

[4] Note, if x is a labelled dictionary and $\mathcal{L}$ returns a dictionary which includes a subset of entries in x and optionally any other additional records, $\mathcal{L}^+$ strictly returns more records than $\mathcal{L}$.

When any registered and corrupted analyst executes an ANALYSE request in the ideal world, the simulator allows the ideal functionality to return the ideal result of the computation only if the adversary instructs the analyst to correctly aggregate the ciphertexts stored in the bulletin board through DD-FESR decryption requests to the appropriate aggregator function. 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 $\mathcal{F}_{\text{EN+}}$ such as $(\text{SETUP}, \epsilon^*)$ and $(\text{FAKEREALITY}, \phi)$ are allowed and redirected to $\mathcal{F}_{\text{EN}}$, as long as $\epsilon^* \in E$ and $\phi \in \Phi$).

Simulator $\mathcal{S}_{\mathsf{GV}}$

State variables	Description
$\mathcal{L}^+$	function to leak anything that $\mathcal{L}$ does, as well as the contents of SEC for all users
$\mathcal{T} \leftarrow \{\}$	Table that stores messages uploaded to the Trusted Bulletin Board
$\widetilde{\mathbf{U}}$	List of corrupted users
SE	List of exposed users

```
On message (SHAREEXPOSURE, $U, u^e_{\mathsf{SEC}}$) from $\mathcal{F}_{\text{EN+}}$:
  simulate sending SETUP to $DD$-FESR on behalf of $U$
  if $u^e_{\mathsf{SEC}} = \perp$ then
      // simulate honest user:
      send LEAK to $\mathcal{F}_{\text{EN+}}$ and receive $\mathsf{r}$
      $u^e_{\mathsf{SEC}} \leftarrow \mathsf{r}[U][\mathsf{SEC}]$
  simulate sending (ENCRYPT, $u^e_{\mathsf{SEC}}, K(|DD\text{-FESR}.\mathbf{A}|)$) to $DD$-FESR through
  $U$ and receive (ENCRYPTED, $\mathsf{h}$)
  $\mathbf{SE} \leftarrow \mathbf{SE} \parallel U$; $\mathcal{T} \leftarrow \mathcal{T} \parallel \mathsf{h}$
On message (REGISTERANALYST, $\alpha$, $\ddot{\mathsf{A}}$) from $\mathcal{F}_{\text{EN+}}$:
  simulate sending SETUP to $DD$-FESR on behalf of $\ddot{\mathsf{A}}$
  for $U \in \mathbf{SE}$ do simulate sending $\Big(\text{REGISTERANALYSTREQUEST}, \alpha, \ddot{\mathsf{A}}, U\Big)$ to
  $\mathcal{Z}$ on behalf of $\ddot{\mathsf{A}}$
On message (REGISTERANALYSTACCEPT, $\alpha, \ddot{\mathsf{A}}, U$) from $\mathcal{F}_{\text{EN+}}$:
  if $U \in \widetilde{\mathbf{U}}$ then
      await for $\Big(\text{KEYSHAREGEN}, \mathsf{AggS}_\alpha, \ddot{\mathsf{A}}\Big)$ from $U$ to $DD$-FESR
  else
      simulate sending $\Big(\text{KEYSHAREGEN}, \mathsf{AggS}_\alpha, \ddot{\mathsf{A}}\Big)$ to $DD$-FESR on behalf
  of $U$
  return OK
On message (ANALYSED, $\alpha, P, \mathrm{y}$) from $\mathcal{F}_{\text{EN+}}$:
  await for (ENCRYPT, $(|\mathcal{T}|, 0)$) from $P$ to $DD$-FESR and for response (ENCRYPTED, $\mathsf{h}_n$)
  await for (DECRYPT, $\mathsf{h}_n, \mathsf{AggS}_\alpha$) from $P$ to $DD$-FESR and for response (DECRYPTED, $|\mathcal{T}|$)
  for $\mathsf{h} \in \mathcal{T}$ do
      await for (DECRYPT, $\mathsf{h}, \mathsf{AggS}_\alpha$) from $P$ to $DD$-FESR and for response (DECRYPTED, _)
```

```
    **return** OK
  *On message* LEAK *from* $\mathcal{A}$ *to* $\mathcal{F}_{\mathrm{EN}}$:
    **send** LEAK **to** $\mathcal{F}_{\mathrm{EN}+}$ and **receive** $\mathsf{r}$
    **return** $\mathcal{L}(\mathsf{r})$
  *On message* $(\textsc{FakeReality}, \phi)$ *from* $\mathcal{A}$ *to* $\mathcal{F}_{\mathrm{EN}}$:
    **assert** $\phi \in \Phi$
    **send** $(\textsc{FakeReality}, \phi)$ **to** $\mathcal{F}_{\mathrm{EN}+}$
  *On message* $(\textsc{Corrupt}, U)$ *from* $\mathcal{A}$ *to* $\mathcal{F}_{\mathrm{EN}}$:
    $\widetilde{\mathbf{U}} \leftarrow \widetilde{\mathbf{U}} \parallel U$
    **send** $(\textsc{Corrupt}, U)$ **to** $\mathcal{F}_{\mathrm{EN}+}$ and **receive** $\overline{A}_U$
    **for** $(\alpha, \ddot{\mathsf{A}}) \in \overline{A}_U$ **do**
        **simulate sending** $\left(\textsc{KeyShareGen}, \mathsf{AggS}_\alpha, \ddot{\mathsf{A}}\right)$ **to** $DD$-FESR **on behalf
    of** $U$
  *On message* * *from* $\mathcal{A}$ *to* $\mathcal{F}_{\mathrm{EN}}$:
    **send** * **to** $\mathcal{F}_{\mathrm{EN}+}$
  *On message* RETRIEVE *from* $\mathcal{A}$ *to* $\mathcal{F}_{\mathsf{TBB}}$:
    **return** $\mathcal{T}$
```

We prove that the Glass-Vault protocol is secure under a model of semi-honest adversarial behaviour; this means showing that $\mathsf{view}^{\mathcal{Z}}_{\mathsf{REAL}} \approx \mathsf{view}^{\mathcal{Z}}_{\mathsf{IDEAL}}$.

We argue that for all messages sent by the environment, the ideal world simulator produces a view that is indistinguishable from $\mathsf{view}^{\mathcal{Z}}_{\mathsf{REAL}}$. In particular:

- SHAREEXPOSURE: when a user U shares their exposure status, $\mathcal{S}_{\mathsf{GV}}$ is activated. If U is corrupted, it additionally receives the noisy record of U's sensitive data, u^{e}_{SEC}. For honest users, $\mathcal{S}_{\mathsf{GV}}$ obtains the same sensitive records by using the LEAK function. As in the real world, DD-FESR is invoked to encrypt the sensitive data, and the resulting handle is stored in an emulated trusted bulletin board. In both worlds, the array of stored handles follows a similar distribution as they are both generated by DD-FESR for the same messages. All other behaviour of SHAREEXPOSURE is handled by $\mathcal{F}_{\mathrm{EN}^+}$ in the same way that $\mathcal{F}_{\mathrm{EN}}$ would.
- REGISTERANALYST: when analyst $\ddot{\mathsf{A}}$ requests permission to compute a function $\alpha \in AF$, the simulator registers them as a decryptor in DD-FESR (the same analyst can request to be registered multiple times, but the DD-FESR functionality will ignore all but the first request). $\mathcal{S}_{\mathsf{GV}}$ then emulates a request for REGISTERANALYSTREQUEST for all exposed users. For the semi-honest case, when both honest and corrupted users are allowed by the environment to accept the request for a function, they will ask DD-FESR for a KEYSHAREGEN. $\mathcal{S}_{\mathsf{GV}}$ learns abouts these REGISTERANALYSTACCEPT calls when either the analyst or user is corrupted. In the former case $\mathcal{S}_{\mathsf{GV}}$ proactively sends the request to DD-FESR on behalf of the user, while in the latter it waits for the adversary to trigger the request. If both user and analyst were honest, the adversary (in either world) should not learn that a request was granted. However, given we are in the dynamic user corruption setting, the simulator has to handle keyshare generation for newly corrupted

users, by requesting keyshare generation to DD-FESR for all functions they had authorised pre-corruption.
- Analyse: for a corrupted analyst, $\mathcal{S}_{\mathsf{GV}}$ will ensure that they sync the state of the ideal function α with that of the aggregated AggS_α in DD-FESR. To aggregate all inputs stored in its emulated trusted bulletin board $\mathcal{T}$, the analyst first encrypts the integer equal to the size of $\mathcal{T}$ and passes it for decryption to AggS_α to initialise it. Then, for all handles stored in $\mathcal{T}$, it also decrypts the corresponding values to the same aggregator. When all the decryptions have occurred, the final returned value will be the evaluation of α on the sensitive data of all exposed users; $\mathcal{S}_{\mathsf{GV}}$ ignores this value and yields back to $\mathcal{F}_{\mathrm{EN}^+}$, which will return the ideal world result to the analyst.
  Since the inputs to the aggregator (the set of uploaded sensitive data to the trusted bulletin board) in the real world fully correspond to the input of α in the ideal, the distributions of states and outputs for AggS_α in DD-FESR and for α in $\mathcal{F}_{\mathrm{EN}^+}$ are indistinguishable.
- Leak: on an adversarial request to learn some values from the combination of noisy record of reality, exposed users, and corrupted users, $\mathcal{S}_{\mathsf{GV}}$ obtains the corresponding leakage r from $\mathcal{F}_{\mathrm{EN}^+}$, and filters it by the admissible leakage $\mathcal{L}$ for $\mathcal{F}_{\mathrm{EN}}$.
- FakeReality: $\mathcal{S}_{\mathsf{GV}}$ ensures that the request to modify the noisy record of reality through $\mathcal{F}_{\mathrm{EN}^+}$ is also admissible in the real world with $\mathcal{F}_{\mathrm{EN}}$.

All other messages are handled by redirecting them from $\mathcal{F}_{\mathrm{EN}^+}$ to $\mathcal{F}_{\mathrm{EN}}$, since both functionalities behave in the same manner outside the cases we have already outlined. $\square$

In the rest of this section, we make a few additional remarks. At a high level, the Glass-Vault protocol can support any computation in a privacy-preserving manner, in the sense that nothing beyond the computation result is revealed to the analyst; more formally, in the simulation-based model, a corrupted party's view of the protocol execution can be simulated given only its input and output. The Glass-Vault protocol can be considered as an *interpreter* that takes a description of any multi-input functionality along with a set of inputs, executes the functionality on the inputs and returns only the result to a potentially semi-honest analyst.

While the above simulator guarantees security in the semi-honest setting, it is possible to design a simulator that allows corrupted parties to diverge from the protocol. In particular, this simulator would need to handle the case of a malicious user who encrypts via DD-FESR dishonestly generated data (in that it does not match with the corrupted user's physical reality measurements). Following the lead of Canetti et al. [19], we can account for these malicious ciphertexts by using functions in Φ^+ to modify the noisy record of physical reality in $\mathcal{F}_{\mathrm{EN}^+}$. Note that this simulation strategy imposes additional deviations from the physical reality beyond those unavoidably inherited by Φ due to its own simulation needs. This makes it harder to justify the usage of the protocol by an analyst who is interested in the correctness of the data processing; hence, our choice of adversarial model.

Recall that the primary reason the Glass-Vault protocol offers “transparency” is that it lets users have a chance to decide which computation should be executed on their sensitive data. This is of particular importance because the result of any secure computation (including functional encryption) would reveal some information about the computation’s inputs. However, having such an interesting feature introduces a trade-off: if many users value their privacy and decide not to share access to their data, the analyst may not get enough to produce any useful results, foregoing the collective benefits this kind of data sharing can engender [38]. One way to solve such a dilemma would be to integrate a (e.g., blockchain-based) mechanism to incentivise users to grant access to their data for such computations; however, even then careful considerations are required, as the framing of why a user is asked to disclose their information can impact how much they value privacy [2].

4.4 Cost evaluation

In the Glass-Vault 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. An analyst’s 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 most efficient EN protocols that realise $\mathcal{F}_{\text{EN}}$ (such as the protocol in [19, Section 8.5]): besides passively collecting measurements of sensitive data, no other data-analytics operation is required until the ShareExposure phase.

While the costliest component of the protocol is the functional encryption module, it is possible to build an efficient implementation of Glass-Vault due to the construction of DD-Steel, which relies on efficient operations facilitated by trusted hardware.

5 Example: infections heatmap

In this section, we provide a concrete example of a computation that a Glass-Vault analyst can 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 defined as 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).

Given the full set of exposed users’ sensitive data, the Heatmap function filters it to the exposed users’ location history for the last $\mathcal{T}$ days (the maximum number of days since they might have been spreading the virus due to its incubation period), and constructs a list of $\mathcal{T} \times k$ matrices, where an entry in each matrix u contains the number of hours within a day an infected individual spent in a particular location. Location data is collected once every hour by the user's phone, and divided into k bins. The u matrix rows are in reverse chronological order, 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, 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 every call with input x, the function allocates a new circular buffer b for each matrix u it constructed from x, and assigns each of u's rows to one of b's $\mathcal{T}$ elements, starting from the top row. Each element in 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 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 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 Supporting material 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.

While the security proof of Theorem 2 is in the semi-honest setting, a fully malicious setting is not unrealistic, as a user can tamper with their own client applications to upload malicious data (terrorist attack [44]). 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 [28]), it is impossible to certify that the users' inputs are valid. Like most other remote computation systems, Glass-Vault cannot provide blanket protection against this kind of attack. However, due to 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 Φ^+ 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 produces similar output. Their scheme relies on combining infection data provided by health authorities with the mass collection of cell phone location data from mobile phone operators. Unlike Glass-Vault, with its 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 and opt-out of the computation.

6 Future directions

There are several possible directions for future research. An immediate goal would be to implement Glass-Vault for various data analytics, examine their run-time, and optimise the system's bottlenecks. Since Glass-Vault's analytics results could influence public policy, it is interesting to investigate how this platform could be equipped with mechanisms that allow result recipients to verify the authenticity of outputs provided by the analyst. Another appealing future research direction is to investigate the design of privacy-preserving contact graph analysis as an application of Glass-Vault, which would let an analyst construct a contact (sub)graph, using users' data, with minimum leakage and maximum transparency, bridging the gap between centralised and decentralised contact tracing.

Acknowledgments

Lorenzo Martinico was supported by the National Cyber Security Centre, the UK Research Institute in Secure Hardware and Embedded Systems (RISE). Aydin Abadi was supported in part by The National Research Centre on Privacy, Harm Reduction and Adversarial Influence Online (REPHRAIN), under UKRI grant: EP/V011189/1.

References

[1] Abbas, R., Michael, K.: COVID-19 contact trace app deployments: learnings from australia and singapore. IEEE Consumer Electronics Magazine **9**(5), 65–70 (2020)

[2] Acquisti, A., John, L.K., Loewenstein, G.: What is privacy worth? The Journal of Legal Studies **42**(2), 249–274 (2013)

[3] Agrawal, S., Goyal, R., Tomida, J.: Multi-party functional encryption. In: Nissim, K., Waters, B. (eds.) 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, pp. 224–255, Springer (2021), URL https://doi.org/10.1007/978-3-030-90453-1_8

[4] Ahmed, N., Michelin, R.A., Xue, W., Ruj, S., Malaney, R.A., Kanhere, S.S., Seneviratne, A., Hu, W., Janicke, H., Jha, S.K.: A survey of COVID-19 contact tracing apps. IEEE Access **8**, 134577–134601 (2020), URL https://doi.org/10.1109/ACCESS.2020.3010226

[5] AISEC, F.: Pandemic contact tracing apps: DP-3T, PEPP-PT NTK, and ROBERT from a privacy perspective. Cryptology ePrint Archive, Report 2020/489 (2020), https://eprint.iacr.org/2020/489

[6] Alliance, U.H.D.R., NHSX: Building Trusted Research Environments - Principles and Best Practices; Towards TRE ecosystems (Dec 2021), URL https://doi.org/10.5281/zenodo.5767586

[7] Alsdurf, H., Bengio, Y., Deleu, T., Gupta, P., Ippolito, D., Janda, R., Jarvie, M., Kolody, T., Krastev, S., Maharaj, T., Obryk, R., Pilat, D., Pisano, V., Prud'homme, B., Qu, M., Rahaman, N., Rish, I., Rousseau, J.F., Sharma, A., Struck, B., Tang, J., Weiss, M., Yu, Y.W.: Covi white paper. CoRR (2020), URL http://arxiv.org/abs/2005.08502v1

[8] Altshuler, T.S., Hershkowitz, R.A.: How Isreal's COVID-19 mass surveillance operation works (2020), https://www.brookings.edu/techstream/how-israels-covid-19-mass-surveillance-operation-works/

[9] Apple, Google: Exposure notification API. https://www.google.com/covid19/exposurenotifications/ (2020)

[10] Avitabile, G., Botta, V., Iovino, V., Visconti, I.: Towards defeating mass surveillance and SARS-CoV-2: The pronto-C2 fully decentralized automatic contact tracing system. Cryptology ePrint Archive, Report 2020/493 (2020), https://eprint.iacr.org/2020/493

[11] Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain UC. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III, LNCS, vol. 12552, pp. 1–30, Springer, Heidelberg, Germany, Durham, NC, USA (Nov 16–19, 2020), https://doi.org/10.1007/978-3-030-64381-2˙1

[12] Bhatotia, P., Kohlweiss, M., Martinico, L., Tselekounis, Y.: Steel: Composable hardware-based stateful and randomised functional encryption. In: Garay, J. (ed.) PKC 2021, Part II, LNCS, vol. 12711, pp. 709–736, Springer, Heidelberg, Germany, Virtual Event (May 10–13, 2021), https://doi.org/10.1007/978-3-030-75248-4˙25

[13] Biasse, J.F., Chellappan, S., Kariev, S., Khan, N., Menezes, L., Seyitoglu, E., Somboonwit, C., Yavuz, A.: Trace-Σ: a privacy-preserving contact tracing app. Cryptology ePrint Archive, Report 2020/792 (2020), https://eprint.iacr.org/2020/792

[14] Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011, LNCS, vol. 6597, pp. 253–273, Springer, Heidelberg, Germany, Providence, RI, USA (Mar 28–30, 2011), https://doi.org/10.1007/978-3-642-19571-6˙16

[15] Bruni, A., Helminger, L., Kales, D., Rechberger, C., Walch, R.: Privately connecting mobility to infectious diseases via applied cryptography. Cryptology ePrint Archive, Report 2020/522 (2020), https://eprint.iacr.org/2020/522

[16] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145, IEEE Computer Society Press, Las Vegas, NV, USA (Oct 14–17, 2001), https://doi.org/10.1109/SFCS.2001.959888

[17] Canetti, R.: Universally composable signature, certification, and authentication. In: 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA, p. 219, IEEE Computer Society (2004), URL https://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24

[18] Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007, LNCS, vol. 4392, pp. 61–85, Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Feb 21–24, 2007), https://doi.org/10.1007/978-3-540-70936-7˙4

[19] Canetti, R., Kalai, Y.T., Lysyanskaya, A., Rivest, R.L., Shamir, A., Shen, E., Trachtenberg, A., Varia, M., Weitzner, D.J.: Privacy-preserving automated exposure notification. Cryptology ePrint Archive, Report 2020/863 (2020), https://eprint.iacr.org/2020/863

[20] Canetti, R., Trachtenberg, A., Varia, M.: Anonymous collocation discovery: Harnessing privacy to tame the coronavirus (2020)

[21] Castelluccia, C., Bielova, N., Boutet, A., Cunche, M., Lauradoux, C., Métayer, D.L., Roca, V.: DESIRE: A third way for a european exposure notification system leveraging the best of centralized and decentralized systems. CoRR **abs/2008.01621** (2020), URL https://arxiv.org/abs/2008.01621

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

[23] Chassang, G.: The impact of the EU general data protection regulation on scientific research. *e*cancermedicalscience **11** (2017)

[24] Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II, LNCS, vol. 11273, pp. 703–732, Springer, Heidelberg, Germany, Brisbane, Queensland, Australia (Dec 2–6, 2018), https://doi.org/10.1007/978-3-030-03329-3˙24

[25] Chotard, J., Dufour-Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Dynamic decentralized functional encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I, LNCS, vol. 12170, pp. 747–775, Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020), https://doi.org/10.1007/978-3-030-56784-2˙25

[26] Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014, LNCS, vol. 8441, pp. 578–602, Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014), https://doi.org/10.1007/978-3-642-55220-5˙32

[27] Gvili, Y.: Security analysis of the COVID-19 contact tracing specifications by apple inc. and google inc. Cryptology ePrint Archive, Report 2020/428 (2020), https://eprint.iacr.org/2020/428

[28] Günther, D., Holz, M., Judkewitz, B., Möllering, H., Pinkas, B., Schneider, T.: Pem: Privacy-preserving epidemiological modeling. Cryptology ePrint Archive, Report 2020/1546 (2020), https://ia.cr/2020/1546

[29] Hassandoust, F., Akhlaghpour, S., Johnston, A.C.: Individuals' privacy concerns and adoption of contact tracing mobile applications in a pandemic: A situational privacy calculus perspective. Journal of the American Medical Informatics Association **28**(3), 463–471 (11 2020), ISSN 1527-974X, URL https://doi.org/10.1093/jamia/ocaa240

[30] Lueks, W., Gürses, S.F., Veale, M., Bugnion, E., Salathé, M., Paterson, K.G., Troncoso, C.: CrowdNotifier: Decentralized privacy-preserving presence tracing. PoPETs **2021**(4), 350–368 (Oct 2021), https://doi.org/10.2478/popets-2021-0074

[31] Mailthody, V.S., Wei, J., Chen, N., Behnia, M., Yao, R., Wang, Q., Agrawal, V., He, C., Wang, L., Chen, L., Agarwal, A., Richter, E., Hwu, W.M., Fletcher, C.W., Xiong, J., Miller, A., Patel, S.: Safer Illinois and Rokwall: Privacy preserving university health apps for Covid-19. CoRR (2021), URL http://arxiv.org/abs/2101.07897v1

[32] Martin, T., Karopoulos, G., Hernández-Ramos, J.L., Kambourakis, G., Fovino, I.N.: Demystifying Covid-19 digital contact tracing: a survey on frameworks and mobile apps. CoRR (2020), URL http://arxiv.org/abs/2007.11687v1

[33] Mascia, C., Sala, M., Villa, I.: A survey on functional encryption. CoRR **abs/2106.06306** (2021), URL https://arxiv.org/abs/2106.06306

[34] Matt, C., Maurer, U.: A definitional framework for functional encryption. Cryptology ePrint Archive, Report 2013/559 (2013), https://eprint.iacr.org/2013/559

[35] Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D.: Location privacy via private proximity testing. In: NDSS 2011, The Internet Society, San Diego, CA, USA (Feb 6–9, 2011)

[36] Pass, R., Shi, E., Tramèr, F.: Formal abstractions for attested execution secure processors. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I, LNCS, vol. 10210, pp. 260–289, Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017), https://doi.org/10.1007/978-3-319-56620-7˙10

[37] Reichert, L., Brack, S., Scheuermann, B.: Ovid: Message-based automatic contact tracing. Cryptology ePrint Archive, Report 2020/1462 (2020), https://eprint.iacr.org/2020/1462

[38] Schwartz, P.M.: Privacy and the economics of personal health care information. Tex. L. Rev. **76**, 1 (1997)

[39] Sturzenegger, D., Sardon, A., Deml, S., Hardjono, T.: Confidential computing for privacy-preserving contact tracing. CoRR (2020), URL http://arxiv.org/abs/2006.14235v1

[40] Tonicelli, R., David, B.M., Alves, V.: Universally composable private proximity testing. In: Boyen, X., Chen, X. (eds.) ProvSec 2011, LNCS, vol. 6980, pp. 222–239, Springer, Heidelberg, Germany, Xi'an, China (Oct 16–18, 2011)

[41] Tramer, F., Zhang, F., Lin, H., Hubaux, J.P., Juels, A., Shi, E.: Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. Cryptology ePrint Archive, Report 2016/635 (2016), `https://eprint.iacr.org/2016/635`

[42] Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy. IEEE Data Eng. Bull. **43**(2), 95–107 (2020), URL `http://sites.computer.org/debull/A20june/p95.pdf`

[43] Troncoso, C., Payer, M., Hubaux, J., Salathé, M., Larus, J.R., Lueks, W., Stadler, T., Pyrgelis, A., Antonioli, D., Barman, L., Chatel, S., Paterson, K.G., Capkun, S., Basin, D.A., Beutel, J., Jackson, D., Roeschlin, M., Leu, P., Preneel, B., Smart, N.P., Abidin, A., Gurses, S., Veale, M., Cremers, C., Backes, M., Tippenhauer, N.O., Binns, R., Cattuto, C., Barrat, A., Fiore, D., Barbosa, M., Oliveira, R., Pereira, J.: Decentralized privacy-preserving proximity tracing. IEEE Data Eng. Bull. **43**(2), 36–66 (2020), URL `http://sites.computer.org/debull/A20june/p36.pdf`

[44] Vaudenay, S.: Centralized or decentralized? The contact tracing dilemma. Cryptology ePrint Archive, Report 2020/531 (2020), `https://eprint.iacr.org/2020/531`

[45] Vaudenay, S., Vuagnoux, M.: The dark side of SwissCovid. `https://lasec.epfl.ch/people/vaudenay/swisscovid` (2020)

[46] Wu, P., Shen, Q., Deng, R.H., Liu, X., Zhang, Y., Wu, Z.: ObliDC: An SGX-based oblivious distributed computing framework with formal proof. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirda, E., Liang, Z. (eds.) ASIACCS 19, pp. 86–99, ACM Press, Auckland, New Zealand (Jul 9–12, 2019), https://doi.org/10.1145/3321705.3329822

A Syntax of Functional Encryption

FE is defined over a class of functions $\mathtt{F} = \{\mathrm{F} \mid \mathrm{F} : \mathcal{X} \to \mathcal{Y}\}$, 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^λ as input and outputs a master keypair $(\mathsf{mpk}, \mathsf{msk})$.
– $\underline{\mathsf{KeyGen}}$ (run by $\mathsf{C} \in \mathbf{C}$). It takes msk and a function's description $\mathrm{F} \in \mathtt{F}$ as inputs and outputs functional key sk_{F}.
– $\underline{\mathsf{Enc}}$ (run by $\mathsf{A} \in \mathbf{A}$). It takes a plaintext string $\mathrm{x} \in \mathcal{X}$ and mpk as inputs. It returns a ciphertext ct or an error.
– $\underline{\mathsf{Dec}}$ (run by $\mathsf{B} \in \mathbf{B}$). It takes ciphertext ct and functional key sk_{F} as inputs and returns a value $\mathrm{y} \in \mathcal{Y}$.

Informally, correctly evaluating the decryption operation on a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathrm{x})$ using functional key sk_{F} should result in $\mathrm{y} \leftarrow \mathrm{F}(\mathrm{x})$. The party B should not learn anything about A's input, except for any information that y reveals about x and 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, EN^{+}, and $\mathsf{Glass\text{-}Vault}$ (presented in Subsections 3.2, 4.1, and 4.2, respectively). The overview is sufficient for the clarification of the communication interface among 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 $\mathcal{F}_{\mathrm{EN}}$, which we present in detail as pseudocode to allow an easy comparison with the extended functionality $\mathcal{F}_{\mathrm{EN}^{+}}$.

B.1 The global attestation functionality G_{att}

The ideal functionality G_{att} was introduced in [36] 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 spk as the master verification key $\mathsf{vk}_{\mathsf{att}}$, available to enclave programs, and ssk as the master secret key msk, protected by the hardware.
- On message GETPK from a party P, it returns $\mathsf{vk}_{\mathsf{att}}$.
- On message $(\text{INSTALL}, \mathsf{idx}, \mathsf{prog})$ from a (registered and honest) party P, it asserts that idx corresponds to the calling party's session id. Then, it creates a unique enclave identifier eid and establishes a software enclave for (eid, P) as $(\mathsf{idx}, \mathsf{prof}, \emptyset)$. It provides P with eid.
- On message $(\text{RESUME}, \mathsf{eid}, \mathsf{input})$ from a (registered) party P, it calls the enclave $(\mathsf{idx}, \mathsf{prog}, \mathsf{mem})$ for (eid, P), where mem is the current memory state. It runs $\mathsf{prog}(\mathsf{input}, \mathsf{mem})$ which returns output and an update memory state mem'. Finally, it produces a signature σ on $(\mathsf{idx}, \mathsf{eid}, \mathsf{prog}, \mathsf{output})$ using 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 Σ, returning the verification key VK on all subsequent calls to $GetK$. In a call to $Sign$, the input is a message $(\textsc{Sign}, \mathsf{vk})$ 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 $(\textsc{send}, m)$ from S, it sends $(\textsc{sent}, m)$ 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 $(\textsc{write}, x)$ from W, it runs the subroutine `getHandle` to obtain an identifying handle h, and records x in $M[\mathsf{h}]$. On message $(\textsc{read}, \mathsf{h})$ 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 $\mathbb{R}$

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 $(\textsc{MyCurrentMeas}, P, L, e)$ 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 $\mathcal{F}_{\mathbf{EN}}$

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 $\mathcal{F}_{\mathrm{EN}^+}$.

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 $\mathcal{F}_{\mathrm{EN}}[\rho, E, \Phi, \mathcal{L}]$

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

On message $(\textsc{Setup}, \epsilon^*)$ *from* $\mathcal{A}$*:*

assert $\epsilon^* \in E$; $\widetilde{\mathrm{R}}_\epsilon \leftarrow \emptyset$

On message $(\textsc{ActivateMobileUser}, U)$ *from a party* P*:*

```
    $\overline{\mathbf{U}} \leftarrow \overline{\mathbf{U}} \parallel U$
    **send** $(\textsc{ActivateMobileUser}, U)$ **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** $\widetilde{\mathrm{R}}_\epsilon[U][\mathsf{INFECTED}] = \bot$ **then**
        **return** error
    **else**
        **send** TIME **to** $\mathbb{T}$ and **receive** $t$
        $\mathbf{SE} \leftarrow \mathbf{SE} \parallel (U, t)$
        $\overline{\mathbf{U}} \leftarrow \overline{\mathbf{U}} \setminus \{U\}$
        **send** $(\textsc{ShareExposure}, U)$ **to** $\mathcal{A}$
  *On message* $(\textsc{ExposureCheck}, U)$ *from a party* $P$*:*
    **if** $U \in \overline{\mathbf{U}}$ **then**
        **send** $(\textsc{AllMeas}, \epsilon^*)$ **to** $\mathbb{R}$ and **receive** $\widetilde{\mathrm{R}}^*$
        $\widetilde{\mathrm{R}}_\epsilon \leftarrow \widetilde{\mathrm{R}}_\epsilon \parallel \widetilde{\mathrm{R}}^*$
        $\mu \leftarrow \widetilde{\mathrm{R}}_\epsilon[U] \parallel \widetilde{\mathrm{R}}_\epsilon[\mathbf{SE}]$
        **return** $\rho(U, \mu)$
    **else return** error
  *On message* $(\textsc{RemoveMobileUser}, U)$ *from a party* $P$*:*
    $\overline{\mathbf{U}} \leftarrow \overline{\mathbf{U}} \setminus \{U\}$
  *On message* $(\textsc{Corrupt}, U)$ *from* $\mathcal{A}$*:*
    $\widetilde{\mathbf{U}} \leftarrow \widetilde{\mathbf{U}} \parallel U$
  *On message* $(\textsc{MyCurrentMeas}, U, A, e)$ *from* $\mathcal{A}$*:*
    **if** $U \in \widetilde{\mathbf{U}}$ **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* LEAK *from* $\mathcal{A}$*:*
    **send** $(\textsc{Leak}, \mathcal{L}(\{\widetilde{\mathrm{R}}_\epsilon, \overline{\mathbf{U}}, \mathbf{SE}\}))$ **to** $\mathcal{A}$
  *On message* $(\textsc{IsCorrupt}, U)$ *from* $\mathcal{Z}$*:*
    **return** $U \overset{?}{\in} \widetilde{\mathbf{U}}$
```

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

The functionality $\mathcal{F}_{\mathsf{TBB}}$, 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 $(\textsc{Add}, c)$ from a party P, it checks with $\mathbb{R}$ whether P is infectious (formally, $\mathcal{F}_{\mathsf{TBB}}$ sends a message $(\textsc{MyCurrentMes}, P,$ "$\mathsf{health\_status}$"$, \mathsf{id})$ 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. For simplicity of exposition, we assume that the input x is already a fully formed list of $\mathcal{T} \times k$ matrices containing a single user's location data over the last $\mathcal{T}$ days. While Glass-Vault functionalities typically expect a subset of $\mathbb{R}$'s noisy record of reality for fields in SEC, turning those records in a list of matrices can be delegated to the aggregator run by Glass-Vault to turn individual user's ciphertext into the multi-input list x.

```
function $\mathsf{Heatmap}_{k,q}(\mathrm{x}, \mathsf{state})$
    if $\mathsf{state} = \emptyset$ then $\mathsf{m} \leftarrow []$
    for $c \in \mathsf{m}$ do
        $c \leftarrow c \parallel \vec{0}$
        if $\forall i \in c : i = \vec{0}$ then $\mathsf{m} \leftarrow \mathsf{m} \setminus c$
    for $u \in \mathrm{x}$ do
        $b \leftarrow \mathsf{CircularBuffer}(\mathcal{T})$
        for $\{i = 0; i < \mathcal{T}; i{+}{+}\}$ do
            assert $\sum_{j=0}^{k-1} u[i, j] = 24$
            $b \leftarrow b \parallel u[i, :]$
        $\mathsf{m} \leftarrow \mathsf{m} \parallel b$
    $\mathrm{y} \leftarrow \vec{0}$
    if $|\mathsf{m}| \geq q$ then
        for $u \in \mathsf{m}$ do
            for $\{i = 0; i < \mathcal{T}; i{+}{+}\}$ do
                $\mathrm{y} \leftarrow \mathrm{y} + u[i, :]$
    return 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

D Steel simulator

We now describe the simulator presented in [12], while adapting the message syntax to fit with the messages sent by DD-FESR and DD-Steel.

Simulator $\mathcal{S}_{\mathrm{FESR}}[\mathsf{PKE}, \Sigma, \mathsf{N}, \lambda, \mathtt{F}]$

State variables	Description
$\mathcal{H}[\cdot] \leftarrow \emptyset$	Table of ciphertext and handles in public repository
$\mathcal{K} \leftarrow []$	List of $\mathsf{prog}_{\mathsf{FE}^{\mathsf{vk}}[\mathrm{F}]}$ enclaves and their $\mathsf{eid}_{\mathsf{F}}$
$\mathcal{G} \leftarrow \{\}$	Collects all messages sent to G_{att} and its response
$\mathcal{B} \leftarrow \{\}$	Collects all messages signed by G_{att}
$(\mathsf{crs}, \tau) \leftarrow \mathsf{N}.\mathcal{S}_1$	Simulated reference string and trapdoor

```
On message $(\textsc{setup}, P)$ from $\mathcal{S}_{DD\text{-FESR}}$:
  if $\mathsf{mpk} = \bot$ then
    $\mathsf{eid}_{\mathsf{KME}} \leftarrow G_{\mathsf{att}}.\mathsf{install}(\mathsf{C.sid}, \mathsf{prog}_{\mathsf{KME}^{\mathsf{vk}}})$
    $(\mathsf{mpk}, \sigma_{\mathsf{KME}}) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{KME}}, \mathrm{init}, \mathsf{crs}, \mathsf{C.sid})$
  if $P = \mathsf{A}$ then
    send $(\textsc{setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}})$ to $\mathcal{SC}_{\mathsf{A}}$
  else if $P = \mathsf{B}$ then
    send $(\textsc{setup}, \mathsf{mpk}, \sigma_{\mathsf{KME}}, \mathsf{eid}_{\mathsf{KME}})$ to $\mathcal{SC}^{\mathsf{C}}_{\mathsf{B}}$ and receive $(\textsc{provision}, \sigma, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD})$
    assert $(\mathsf{C.sid}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{vk}}}, \mathsf{pk}_{KD}) \in \mathcal{B}[\sigma]$
    $(\mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}}) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathsf{KME}}, (\mathrm{provision}, (\sigma, \mathsf{eid}_{\mathsf{DE}}, \mathsf{pk}_{KD}, \mathsf{eid}_{\mathsf{KME}}, \mathsf{crs}))))$
    send $(\textsc{provision}, \mathsf{ct}_{\mathsf{key}}, \sigma_{\mathsf{KME}})$ to $\mathcal{SC}_{\mathsf{B}}$
On message $(\textsc{read}, \mathsf{h})$ from party $\mathsf{B}$ to $\mathcal{REP}$:
  send $(\textsc{decrypt}, \mathrm{F}_0, \mathsf{h})$ to $\mathcal{S}_{DD\text{-FESR}}$ on behalf of $\mathsf{B}$ and receive $(\textsc{decrypted}, |(\mathrm{m}, k)|)$
  assert $|(\mathrm{m}, k)| \neq \bot$
  $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{mpk}, 0^{|(\mathrm{m}, k)|})$
  $\pi \leftarrow \mathsf{N}.\mathcal{S}_2(\mathsf{crs}, \tau, (\mathsf{mpk}, \mathsf{ct}))$
  $\mathsf{ct}_{\mathsf{msg}} \leftarrow (\mathsf{ct}, \pi)$; $\mathcal{H}[\mathsf{ct}_{\mathsf{msg}}] \leftarrow h$
  send $(\textsc{read}, \mathsf{ct}_{\mathsf{msg}})$ to $\mathsf{B}$
On message $(\textsc{install}, \mathsf{idx}, \mathsf{prog})$ from party $P \in \{\mathbf{B} \cup \mathsf{C}\}$ to $G_{\mathsf{att}}$:
  $\mathsf{eid} \leftarrow G_{\mathsf{att}}.\mathsf{install}(\mathsf{idx}, \mathsf{prog})$
  $\mathcal{G}[\mathsf{eid}].\mathsf{install} \leftarrow (\mathsf{idx}, \mathsf{prog}, P)$
  // $\mathcal{G}[\mathsf{eid}].install[1]$ is the program's code
  forward eid to B
On message $(\textsc{resume}, \mathsf{eid}, \mathsf{input})$ from party $P \in \{\mathbf{B} \cup \mathsf{C}\}$ to $G_{\mathsf{att}}$:
  assert $\mathcal{G}[\mathsf{eid}].\mathsf{install}[2] = P$
  if $\mathcal{G}[\mathsf{eid}].\mathsf{install}[1] \neq \mathsf{prog}_{\mathsf{FE}^{\mathsf{vk}}[\cdot]} \vee (\mathsf{input}[0] \neq \mathrm{run} \vee \mathsf{input}[-1] \neq \bot)$ then
    $(\mathsf{output}, \sigma) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}, \mathsf{input})$
    $\mathcal{G}[\mathsf{eid}].\mathsf{resume} \leftarrow \mathcal{G}[\mathsf{eid}].\mathsf{resume} \parallel (\sigma, \mathsf{input}, \mathsf{output})$
    $\mathcal{B}[\sigma] \leftarrow (\mathcal{G}[\mathsf{eid}].\mathsf{install}[0], \mathsf{eid}, \mathcal{G}[\mathsf{eid}].\mathsf{install}[1], \mathsf{output})$
    forward $(\mathsf{output}, \sigma)$ to $P$
  else
    $(\mathsf{idx}, \mathsf{prog}_{\mathsf{FE}^{\mathsf{vk}}[\mathrm{F}]}, P) \leftarrow \mathcal{G}[\mathsf{eid}].\mathsf{install}$
    $(\mathrm{run}, \sigma_{\mathsf{DE}}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{ct}_{\mathsf{key}}, \mathsf{ct}_{\mathsf{msg}}, k_{\mathrm{F}}, \mathsf{crs}, \bot) \leftarrow \mathsf{input}$
    assert $(\sigma_{\mathrm{F}}, (\mathrm{init}, \mathsf{mpk}, \mathsf{idx}), (\mathsf{pk}_{\mathsf{FD}})) \in \mathcal{G}[\mathsf{eid}].\mathsf{resume}$
    assert $(\mathsf{idx}, \mathsf{eid}, \mathsf{prog}_{\mathsf{FE}^{\mathsf{vk}}[\mathrm{F}]}, \mathsf{pk}_{\mathsf{FD}}) \in \mathcal{B}[\sigma_{\mathrm{F}}]$
    assert $(\mathsf{idx}, \mathsf{eid}_{\mathsf{DE}}, \mathsf{prog}_{\mathsf{DE}^{\mathsf{vk}}}, \mathsf{ct}_{\mathsf{key}}, k_{\mathrm{F}}, \mathsf{crs})) \in \mathcal{B}[\sigma_{\mathsf{DE}}]$
    // If the ciphertext was not computed honestly and saved to $\mathcal{H}$
    if $\mathcal{H}[\mathsf{ct}_{\mathsf{msg}}] = \bot$ then
```

```
        $(\mathsf{ct}, \pi) \leftarrow \mathsf{ct}_{\mathsf{msg}}$
        $((\mathrm{m}, k), \mathrm{r}) \leftarrow \mathsf{N}.\mathcal{E}(\tau, (\mathsf{mpk}, \mathsf{ct}), \pi)$
        **send** $(\textsc{encrypt}, \mathrm{m}, k)$ **to** $\mathcal{S}_{DD\text{-FESR}}$ on behalf of $P$ and **receive** $(\textsc{encrypted}, \mathsf{h})$
        **if** $\mathsf{h} \neq \perp$ **then** $\mathcal{H}[\mathsf{ct}_{\mathsf{msg}}] \leftarrow \mathsf{h}$
        **else return**
    $\mathsf{h} \leftarrow \mathcal{H}[\mathsf{ct}_{\mathsf{msg}}]$
    **send** $(\textsc{decrypt}, \mathrm{F}, \mathsf{h})$ **to** $\mathcal{S}_{DD\text{-FESR}}$ on behalf of $P$ and **receive** $(\textsc{decrypted}, \mathrm{y})$
    $((\mathrm{computed}, \mathrm{y}), \sigma) \leftarrow G_{\mathsf{att}}.\mathsf{resume}(\mathsf{eid}_{\mathrm{F}}, (\mathrm{run}, \perp, \perp, \perp, \perp, \perp, \perp, \mathrm{y}))$
    $\mathcal{G}[\mathsf{eid}].\mathsf{resume} \leftarrow \mathcal{G}[\mathsf{eid}].\mathsf{resume} \parallel (\sigma, \mathsf{input}, (\mathrm{computed}, \mathrm{y})))$
    $\mathcal{B}[\sigma] \leftarrow (\mathcal{G}[\mathsf{eid}].\mathsf{install}[0], \mathsf{eid}, \mathcal{G}[\mathsf{eid}].\mathsf{install}[1], (\mathrm{computed}, \mathrm{y}))$
    **forward** $((\mathrm{computed}, \mathrm{y}), \sigma)$ to $P$
```