# CoVault: A Secure Analytics Platform

Roberta De Viti[1], Isaac Sheff[1], Noemi Glaeser[2,4], Baltasar Dinis[3], Rodrigo Rodrigues[3], Jonathan Katz[4], Bobby Bhattacharjee[4], Anwar Hithnawi[5], Deepak Garg[1], and Peter Druschel[1]

[1]Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus
[2]Max Planck Institute for Security and Privacy (MPI-SP)
[3]Instituto Superior Tecnico (ULisboa), INESC-ID
[4]University of Maryland
[5]ETH Zürich

***Abstract*—In a *secure analytics platform*, data sources consent to the *exclusive use* of their data for a pre-defined set of analytics queries performed by a specific group of analysts, and for a limited period. If the platform is secure under a sufficiently strong threat model, it can provide the missing link to enabling powerful analytics of sensitive personal data, by alleviating data subjects' concerns about leakage and misuse of data. For instance, many types of powerful analytics that benefit public health, mobility, infrastructure, finance, or sustainable energy can be made differentially private, thus alleviating concerns about *privacy*. However, no platform currently exists that is sufficiently *secure* to alleviate concerns about *data leakage and misuse*; as a result, many types of analytics that would be in the interest of data subjects and the public are not done.**

**CoVault uses a new multi-party implementation of functional encryption (FE) for secure analytics, which relies on a unique combination of secret sharing, multi-party secure computation (MPC), and different trusted execution environments (TEEs). CoVault is secure under a very strong threat model that tolerates compromise and side-channel attacks on any one of a small set of parties and their TEEs. Despite the cost of MPC, we show that CoVault scales to very large data sizes using map-reduce based query parallelization. For example, we show that CoVault can perform queries relevant to epidemic analytics at scale.**

## 1. Introduction

There is a growing need for analytics of sensitive data, such as personal mobility, activity, health, financial, and contact data. Examples include analytics to support the study of epidemics with high spatio-temporal resolution, to optimize sustainable transportation systems, and to aid urban planning; analytics of individual health records at large scale to extract information about rare disease diagnosis and treatment; and analytics of individual financial transactions to extract economic insight or to detect systemic vulnerabilities in the finance system. In these scenarios, analytics that would be of significant societal value are currently not done in many countries due to concerns about privacy and leakage or misuse of data [6], [36]. We contend that a platform with a sufficiently high level of security to enable such analytics would be valuable, even if it required significant compute resources. Such a platform would also promote the availability and safe use of protected public-sector data for research and the public good, a central goal of the EU's new Data Governance Act (DGA) [4].

A *secure data analytics platform* supports the execution of a pre-defined set of analytics queries while preventing direct data access and unauthorized queries. The need for secure analytics arises when the raw data is sensitive, while the results of (statistical) analytics queries are not sensitive but highly valuable to data subjects or the public. In this scenario, data subjects/owners may be willing to contribute their data for specific analytics if, and only if, they have strong assurances that the query results do not violate their privacy, and their data cannot be leaked or used in any other way.

**Challenge.** The design of a secure analytics platform involves trading off the strength of the threat model, the expressiveness of analytics queries, and the efficiency of query processing (i.e., query latency, bandwidth, and resource requirements). In general, a stronger threat model requires more complex security primitives to protect the data, which in turn require more resources to perform analytics. Depending on the scalability of a design, these additional requirements may translate into long query latency or additional storage and compute resources. In many cases restricting query expressiveness can increase efficiency.

Another trade-off concerns the degree of data aggregation. At one extreme, data is aggregated by a single party, which is efficient but challenging to secure; at the other extreme, data remains in the hands of its owners, which trivially prevents data leakage and misuse but limits the expressiveness of queries that can be executed efficiently due to communication and availability limits since a significant percentage of owners must participate in every query.

**Prior work.** Existing work on secure databases and analytics platforms has explored different points in this design space. First, *hardware-secured systems* rely on a Trusted

Execution Environment (TEE) [53], [73], [13], [11], [18], [68], [40], [64], [51], [52]. These systems execute arbitrary queries on encrypted data at near native speed of the underlying compute platform, but their threat model excludes compromise of the TEE or its vendor, as well as side channels; many such attacks have been reported [56]. Second, *federated analytics* systems support differentially private analytics queries over distributed, encrypted data stores [17], [66], [67], [65]. In these systems, data remains under the control of its owners, which avoids many threats associated with aggregating data from different owners by a third party. However, network limitations (delay, bandwidth), availability of data stores, and the overhead of cryptographic primitives required to prevent data leaks via intermediate query results limit the expressiveness of queries. Third, some systems support analytics queries over *homomorphically encrypted (HE)* data [65], [22], [26], [63], [23], [60]. The expressiveness of queries is limited only by the set of homomorphic operations supported by the encryption scheme. Fully homomorphic (FHE) schemes, which support arbitrary computation, are generally still very expensive [72]. Also, query results are encrypted with the same key as the raw data, and therefore a party must be trusted to decrypt the results of approved queries, which requires additional security mechanisms beyond HE [21], [65], [67], [66]. In summary, the setup and trust assumptions of prior work are not fit for the class of analytics we target, the confidentiality guarantees we require, and the user control on data use we wish to impose.

**CoVault.** This paper explores a different design based on two complementary ideas. First, data is entrusted in secret-shared form to a small set of independent parties that store one data share each and jointly process queries using secure multi-party computation (MPC). By hosting the parties in a single data or co-location center, CoVault combines low-latency data access with multi-party security, thereby enabling complex queries while avoiding trust in any one party and secure hardware technology. Second, the parties jointly implement a form of *functional encryption (FE)*. FE is a generalization of public-key encryption where possessing a secret key allows one to learn a pre-defined set $F$ of functions of the encrypted cleartext $m$, but nothing else about $m$. We observe that FE provides the essential functionality of a secure analytics platform. Data sources can encrypt their data using the public key, and $F$ models the set of queries allowed on the data. As a consequence, only the results of pre-approved queries can be obtained from the encrypted data. Unfortunately, existing implementations of FE are limited to specific functions, are impractically expensive, or rely solely on hardware security [8], [5], [39].

**FE construction.** At the heart of CoVault's design is a construction of FE that combines several types of TEEs, secret sharing, and MPC. The construction is secure under a strong threat model that tolerates side channels as well as compromise of any $t$ of $n$ parties, components, or technologies, where $t$ and $n$ depend on the specific MPC protocol used (1 of 2 in our prototype). Briefly, (i) the data sources use $n$-out-of-$n$ secret-sharing on their data, generating $n$ shares; then, they encrypt each share with a different public key $K_{Fi}^{pub}$, where $F$ corresponds to a class of analytics queries to which the data sources consent, and $i$ indicates the share; finally, they send each encrypted share to a separate party within CoVault; (ii) each party processes data within a TEE of a different type and vendor (e.g., Intel SGX, Intel TDX, AMD SEV-SNP, ARM CCA [30], [70], [7], [69]); a TEE obtains access to the private key $K_{Fn}^{priv}$ needed to decrypt its share iff it was attested to implement precisely the query class $F$; (iii) to process a query $f$ (from the set $F$) on the database $m$, the TEEs of each party jointly perform MPC in the malicious threat model on their shares. Each TEE obtains a share of the query result $f(m)$ and reveals it to authorized queriers, who can assemble the shares to obtain $f(m)$.

Intuitively, this construction implements FE in the threat model described above. In fact, (a) only TEEs implementing $f$ can decrypt the data $m$; (b) side-channel attacks against, or compromise of, any $t$ or fewer TEEs are unproductive, as individual TEEs learn only their own share of $m$ plus their share of $f(m)$; (c) secret sharing ensures that any $n-1$ shares reveal nothing about the underlying data; and (d) correlated attacks against TEEs are difficult because each TEE is of a different type and vendor.

**Optimizations.** CoVault's FE construction occupies a middle ground: It is more efficient than purely cryptographic constructions and supports more general queries than federated analytics, but the use of MPC makes it more compute-intensive than systems that rely on near-native query execution within a TEE [53], [73], [13], [11], [18], [68], [40], [64], [51], [52], although the latter systems do not tolerate side channels or compromise of the TEE hardware or vendor. To ensure scalability and performance at scale, CoVault uses a number of optimizations. In order to scale out and support large datasets and complex queries, CoVault relies on map-reduce style distributed processing. Each mapper and reducer is implemented as an MPC instance, with each party in the MPC encapsulated in a TEE of a different type.

Since CoVault's threat model allows side channels, CoVault must not leak secrets indirectly via observable behavior such as memory access, network communication, and database access patterns. We use circuit-based MPC, which has oblivious memory accesses, and CoVault pads all network communication to hide the shape of its payload. To make database accesses oblivious while avoiding full scans for efficiency, CoVault uses several techniques. First, if queries select data by a public attribute, then only the data matching the selection needs to be fetched. Second, CoVault includes an efficient private information retrieval (PIR) protocol for data accesses as part of queries that select data based on private attributes, or perform data-dependent data accesses. Finally, CoVault can produce materialized views during its ingress processing to avoid expensive operations like joins during interactive query processing.

**Results/Limitations.** CoVault is the first analytics platform based on a practical FE construction that is fit for purpose

and secure under a very strong threat model. Thus, CoVault can enable analytics on sensitive data, and it scales out to large datasets using map-reduce parallelism. For instance, § 7 shows that CoVault can perform epidemiologically-relevant analytics on personal mobility and contact data at scale. The CoVault prototype uses garbled circuits, i.e., $n = 2, t = 1$. This choice was motivated in part by the limited number TEE implementations available today, although additional vendors have announced TEEs. Assuming the availability of more TEEs, MPC protocols with $n > 2$ can potentially offer even better efficiency and security, but the details remain for future work.

**Contributions.** (1) A multi-party functional encryption construction in a strong threat model that tolerates side channels and compromise of any $t$ of $n$ TEE implementations. (2) The design, implementation, and evaluation of an analytics platform that is secure under the same threat model. (3) Several technical components: A provably secure secret-sharing scheme with built-in MACs (§ 4.4), a modified 2-party MPC (2PC) protocol that produces asymmetric outputs (§ 4.3), and a simple PIR scheme that supports fetches in constant time (§ 6.1). (4) A data-oblivious map-reduce in 2PC for scalable query processing (§ 6.2).

## 2. CoVault Overview

In this section, we describe the requirements for a secure analytics platform and survey relevant available technologies.

### 2.1. System requirements

Secure analytics platforms require strong technical safeguards to prevent data theft, leaks, manipulation, and misuse, including unauthorized queries. In particular:

**Transparency** The system is fully transparent in its design, implementation (source code), configuration, and operation.

**Decentralized trust** Trust in the system does not depend on any individual party or technology.

**Integrity** Data stored cannot be altered by platform components or external third parties without detection.

**Confidentiality** Subject to a strong threat model (§ 3.2), it is impossible to extract information from the database except through the approved analytics query interface.

**Consent** Users explicitly consent to their data being used for specific and publicly-defined analytics queries, by specific parties, and for a limited period of time. Changes to these categories require renewed and explicit user consent.

**Privacy** While privacy requires query-specific mechanisms and cannot be provided by an analytics platform itself, the system should be compatible with strong privacy-preserving mechanisms like differential privacy.

### 2.2. Building blocks

**Secret sharing** is a method for sharing a secret value among multiple parties, such that the value can be obtained only if a sufficient fraction of shares (possibly all) are combined. CoVault uses secret sharing to tolerate the failure of one class of system components, such as TEEs from a particular vendor, without risking data disclosure.

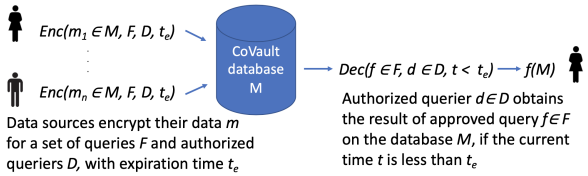
**Secure multi-party computation (MPC)** allows several parties to jointly compute a function without revealing their respective inputs. MPC protocols are secure in either a semi-honest [80], [47] or a malicious [77] threat model. To evaluate queries on secret-shared data, CoVault uses oblivious MPC in the $t$-of-$n$ malicious threat model, where up to $t$ of the $n$ parties may be malicious. The value of the maliciousness threshold $t$ varies from $n/3$ to $n-1$ depending on the protocol. Garbled circuits (GCs) are a specific 2-party MPC (2PC) technique ($n = 2, t = 1$), which is practical in many applications [62]. GCs are inherently oblivious as circuits have no control flow constructs.

**Trusted execution environments (TEE)** are supported by many recent general-purpose CPUs, e.g., Intel SGX, AMD SEV, and the upcoming Intel TDX and ARM CCA [30], [7], [70], [69]. TEEs provide confidentiality and integrity of data and computation under a threat model that tolerates compromised operating systems, hypervisors, and even some physical attacks [24], [30]. Unlike the first-generation SGX, the newer SEV-SNP, TDX, and CCA encapsulate an entire VM, providing a familiar, general programming model and a large memory space. An attestation protocol generates a cryptographic proof that a VM executes in an authentic TEE of a given type and that its initial memory state matches the expected secure hash value (measurement). Given this proof, the VM is provided with cryptographic material needed to authenticate itself to remote parties and to access data sealed for it. CoVault uses TEEs to encapsulate each party's computation. The security guarantees of a TEE depend on the integrity of its vendor's certificate chain, proprietary hardware, and firmware. To minimize the chance of correlated compromises, CoVault relies on multiple TEE implementations from different vendors.

**Functional Encryption (FE)** is a generalization of public-key encryption, which enables the holder of a private key to learn a pre-defined function of what the ciphertext is encrypting, but nothing else [20]. Iron [39] implements FE using a TEE; this construction is efficient but is vulnerable to side channels, and its security relies on the integrity of the TEE's hardware implementation and its vendor. CoVault relies on a new FE-like construction, which extends Iron to be secure without relying on any one TEE implementation or vendor, and tolerates side channels.

**Private Information Retrieval (PIR)** techniques allow a client to fetch data from a database without disclosing *which* data is fetched [29]. CoVault uses PIR to fetch data from materialized views when executing data-dependent queries.

Figure 1: CoVault: Using FE for secure analytics




### 2.3. Approach and roadmap

In CoVault, data sources encrypt their data using a variant of FE, such that only the results of a pre-defined set of queries $F$ can be obtained from the encrypted data, and only by a pre-defined set of authorized queriers $D$, before expiration time $t_e$ (Figure 1). At the heart of CoVault is a multi-party FE-like construction, which we develop through successive refinement of strawman designs in § 3 and fully describe in § 4. In § 5, we describe the design of the CoVault analytics platform based on this FE-like construction, and in § 6 we show how to execute general analytic queries in an efficient and scalable manner. Finally, in § 7 we evaluate CoVault in the context of epidemic analytics as an example application scenario.

## 3. CoVault's FE-like construction

In this section, we present the high-level API of CoVault's FE-like construction, and then sketch the construction by incrementally refining a series of strawman designs.

### 3.1. API

FE allows a party with access to only a ciphertext and an appropriate decryption key to compute a specific function of the cleartext. More precisely, given a set $X$ (plaintext space) and a set $\mathcal{F}$ of functions over $X$ (function space), a FE scheme over $(X, \mathcal{F})$ enables one to evaluate $f(x)$ given the encryption of $x \in X$ and a decryption key $sk_f$ for $f \in \mathcal{F}$ [20]. One cannot learn anything about $x$ other than $f(x)$ using the decryption key $sk_f$.

We use a variant of FE that allows encryptors to specify which functions from the function space can be run on their data, which decryptors can request the evaluation of these functions, and when their data expires. We reflect this variation in the $(Setup, KeyGen, Enc, Dec)$ API, which defines a FE scheme in the literature.

$mpk[F, D, t_e], msk[F, D, t_e] \leftarrow Setup(1^\kappa, F, D, t_e)$ This operation takes as input a security parameter $\kappa$, a set of functions $F$, a set of authorised decryptors $D$ and an expiration time $t_e$. It outputs a master public-private keypair constrained to $F$, $D$ and $t_e$. The public key, $mpk[F, D, t_e]$, is published for use by encryptors, while the private key, $msk[F, D, t_e]$, is held secret by a trusted party.

$sk_f \leftarrow KeyGen(msk[F, D, t_e], f, d)$ This operation takes as input a constrained master secret key $msk[F, D, t_e]$, a specific function $f$, and a decryptor $d$. It outputs a function secret key $sk_f$ if $f \in F$ and $d \in D$. Otherwise, it fails and outputs $\perp$. $sk_f$ can be used to obtain $f(m)$ for any $m$ encrypted using $mpk[F, D, t_e]$ (see $Dec()$ below). $KeyGen()$ is executed by the trusted party on behalf of a decryptor $d$. The decryptor provides $f$, while the trusted party provides $msk[F, D, t_e]$.

$c \leftarrow Enc(mpk[F, D, t_e], m)$ This operation takes as input a constrained master public key $mpk[F, D, t_e]$ and a message $m$. It outputs a ciphertext $c$.

$f(m) \leftarrow Dec(sk_f, c)$ This operation takes as input a function secret key $sk_f$ and a ciphertext $c$. It outputs $f(m)$ if $sk_f \neq \perp$, $c$ was generated using $Enc(mpk[F, D, t_e], m)$, $sk_f$ was generated using $KeyGen(msk[F, D, t_e], f, d)$ with the same $F$, $D$ and $t_e$, and the current time is less than the expiration time $t_e$. Otherwise, it fails and outputs $\perp$.

### 3.2. Desired properties and threat model

**Properties.** We are interested in two security properties:

- **(Data) Confidentiality.** Plaintexts remain confidential, except that authorized decryptors (in the set $D$) may learn authorized functions (from the set $F$) of the plaintexts if they execute $Dec()$ before the expiration time $t_e$. No other entity learns anything about plaintexts.
- **(Result) Integrity.** The decryptor receives the correct value of the its provided function, $f$, applied to the encrypted plaintext, $m$. An adversary cannot modify $f$, $m$, or the result $f(m)$ without detection.

**Threat model.** We aim to provide confidentiality and integrity in the presence of active attacks that may exploit software or hardware vulnerabilities, including side channels. We do not trust any single technology or vendor to produce hardware resilient to strong attacks and, therefore, divide trust among multiple parties and hardware technologies. Specifically, both our FE-like construction and the CoVault design (which builds on this construction) secret-share data among $n$ independent parties for confidentiality, and use $n$-party MPC in the $t$-of-$n$ malicious model to answer queries. To raise the bar for compromise or malice, we encapsulate each party in a TEE of a different vendor and the code of the parties running in the TEEs is attested to be correct. Moreover, we assume that no more than $t$ parties collude and their TEEs are not subject to *simultaneous* side-channel or physical attacks [56]. To make sure a malicious party can attack at most its own TEE in this way, the parties' compute platforms are assumed to be physically separated, e.g., in separate cages of a co-location center.

Our precise assumption is that at least $n-t$ of the parties and their TEEs remain uncompromised and free from side-channel exploits. The remaining $t$ TEEs may be arbitrarily malicious: they may be subject to software or hardware attacks (even in collusion with a hardware vendor) and may run the wrong code (different from what was attested). Our design provides confidentiality and integrity in this very strong threat model.

| | Confidentiality if: | Integrity if: |
|---|---|---|
| S1 | At least $n-t$ parties trusted and free of side-channel exploits | All parties semi-honest (no active attacks) |
| S2 | At least $n-t$ parties with uncompromised TEEs and free of side-channel exploits | All parties semi-honest (no active attacks) |
| S3 | At least $n-t$ parties with uncompromised TEEs and free of side-channel exploits. (Same as our threat model.) | |

Figure 2: Assumptions needed to attain confidentiality and integrity in our Strawman designs

The prototype implementation of our FE-like construction (§ 4) uses two parties ($n = 2$) of which one may be compromised ($t = 1$). For the 2-party MPC (2PC) protocol, we rely on DualEx [48]. Compared to other maliciously-secure 2PC protocols, DualEx is more efficient but may leak one bit of information if the protocol aborts due to an active attack on one of the two parties. This leak cannot be amplified by repetition: DualEx informs the honest party of the attack, so the honest party refuses to interact further with the malicious party. We admit this 1-bit leak tacitly as it provides significant performance gains in return. A different implementation could use a slower 2PC protocol without this 1-bit leak.

Denial-of-service attacks are out of scope; they can be addressed with orthogonal techniques.

Next, we describe three strawman designs that attain confidentiality and integrity under progressively weaker assumptions. The last design matches our threat model and is the basis of our actual FE-like construction (§ 4). Figure 2 summarizes the assumptions of the three designs.

### 3.3. Strawman 1 (S1): *n*-party FE construction

Our first strawman design, S1, implements the FE API of § 3.1 by combining secret sharing and $n$-party MPC. Specifically, $n$ independent parties [1] *jointly* implement the trusted party of FE that holds the master secret key, $msk[F, D, t_e]$.

In S1, confidentiality relies on at least $n-t$ parties being honest and free of side-channel exploits.

**Components.** Encryptors encode data using a $n$-of-$n$ *secret-sharing scheme*, such that the cleartext cannot be recovered unless all of the $n$ shares are available; then, they entrust each data share to a different one of $n$ parties. To decrypt data, the parties run $n$-party MPC on their shares to produce the result $f(m)$. The chosen MPC scheme is secure in the $t$-of-$n$ malicious model: no party learns anything about $m$ other than $f(m)$ as long as at least $n - t$ parties are honest (i.e., they follow the protocol, do not disclose any information not required by the protocol to any other party, and are not subject to any hardware or software attack). Furthermore, the chosen MPC scheme is data oblivious, i.e., without any control flow. A simple way to attain data obliviousness is to use a circuit-based MPC scheme.

1. We do not use the terms "party" or "parties" to refer to other actors in the system like encryptors and decryptors.

**Implementation.** Next, we sketch how S1 implements the operations in § 3.1, which can be run after the $n$ parties are initialized. In contrast with the standard FE API, some elements, which we denote with capital letters, are vectors.

$MPK[F, D, t_e], MSK[F, D, t_e] \leftarrow Setup(1^\kappa, F, D, t_e)$
Each party runs this function locally to produce its own key pair and link it to $F$, $D$ and $t_e$. The output values MPK and MSK are $n$-element vectors, one for each party/share. Each party keeps its *secret* key locally.

$SK_f \leftarrow KeyGen(MSK[F, D, t_e], f, d)$ The decryptor $d$ calls this function on each party and provides the parameter $f$. Regarding the first parameter, $MSK[F, D, t_e]$, each party provides its own secret key. Each party runs the function locally, and independently produces a cryptographic authorization token $sk_f$, linked to $f$ and $t_e$ iff $f \in F$ and the caller $d \in D$. Each party returns its $sk_f$ to the decryptor: the return parameter $SK_f$ represents an $n$-element vector with one authorization token per party.

$C \leftarrow Enc(MPK[F, D, t_e], m)$ Every encryptor runs this function locally. The encryptor secret-shares $m$ into $n$ shares, and encrypts each share with the corresponding party's public key in the vector $MPK[F, D, t_e]$. $C$ represents the vector of encrypted shares returned to the encryptor who may then transmit it to decryptors over any channel.

$f(m) \leftarrow Dec(SK_f, C)$ The decryptor $d$ calls this function on each party, providing the authorization token (previously received from a $KeyGen$ call) and the share encrypted with that party's public key. If each party accepts its authorization token as valid and each party's current clock time is less than $t_e$ (which each party linked with its token), then the parties decrypt their shares using their corresponding secret key of $MSK[F, D, t_e]$, and perform MPC to compute the result $f(m)$ encrypted with the caller's (decryptor's) public key. If any party rejects its authorization token from $SK_f$ or believes $t_e$ is past, the result is $\perp$.

**Properties.** The S1 FE construction provides confidentiality as long as at least $n-t$ parties are honest and free of side-channel exploits (all others may be arbitrarily malicious). Since S1 does not run parties in TEEs (that is covered in S2 below), honesty of the $n-t$ parties assumes the absence of malicious intent as well as hardware and software attacks against those parties. Simultaneous side-channel attacks on all but the $n-t$ honest parties do not violate confidentiality, since such attacks reveal at most $n - t$ of the $n$ shares outside MPC, and nothing from within the MPC (as the MPC scheme is data oblivious). Furthermore, the integrity of $f(m)$ holds as long as all parties are semi-honest and do not manipulate their shares.

### 3.4. Strawman 2 (S2): S1 + TEE encapsulation

Our next strawman design, S2, weakens the assumption of at least $n - t$ honest parties for the *confidentiality* of S1 to the assumption of at least $n - t$ *uncompromised TEE implementations*.

**Components.** S2 additionally executes each of the $n$ parties inside a TEE of a different type. The code of every party is attested to be correct.

**Implementation.** The system initialization involves starting and attesting the TEEs encapsulating each of the $n$ parties. Once the TEEs are started, all communication among and with the TEEs occurs via secure channels tied to the attestation. The API functions are implemented as in S1, but the $n$ parties run in their respective TEEs.

**Properties.** S2 improves S1 by providing confidentiality as long as at least $n - t$ TEEs are uncompromised and free of side-channel exploits. The remaining $t$ parties and their underlying TEEs may be arbitrarily compromised or subject to side-channel exploits. Furthermore, the integrity of $f(m)$ holds as long as all parties are semi-honest and, in particular, do not manipulate their input shares (as in S1).

### 3.5. Strawman 3 (S3): S2 + MACs

Strawman S3 weakens the assumption needed for the *integrity* of $f(m)$ to at least $n - t$ *uncompromised TEE implementations*.

**Components.** S3 is like S2, but adds message authentication codes (MACs) to data shares.

**Implementation.** Same as S2, except that $Enc()$ adds a MAC to each encrypted share. The MPC within a $Dec()$ operation checks these MACs, and produces a MAC on the result $f(m)$, which the caller (decryptor) should check.

**Properties.** S3 provides confidentiality under the same assumptions in S2: At least $n - t$ parties' TEEs must be uncompromised and free of side-channel exploits. Differently from S2, in S3 the integrity of $f(m)$ holds under the same assumption as for confidentiality. Hence, S3 attains confidentiality and integrity in our desired threat model (§ 3.2).

## 4. Details of the FE construction

CoVault's full FE-like construction follows the S3 design. The construction uses two parties, 2PC, and two types of TEEs ($n = 2$ and $t = 1$). The S3 strawman omits many details, which we describe in this section. First, we discuss the mechanisms behind the attestation of TEEs. Then, we present our secret-sharing scheme, and discuss how we implement 2PC using garbled circuits in the malicious threat model by adapting the existing DualEx protocol [48] and adding output integrity guarantees.

### 4.1. TEE attestation

**Background.** Each party is encapsulated in a different TEE implementation, and runs each component of its processing pipeline in a separate TEE of its given implementation. Each party's processing pipeline consists of two kinds of components: a single provisioning service (PS) and one or more data processors (DPs). The PSs perform actions on behalf of their party, and also attest and provision the DPs in their pipeline with the keypair necessary to decrypt shares. The DPs of a party are the entities that jointly perform 2PC with the DPs of the other party: each party's DP represents a *party* in S3, in the sense of 2PC. Each pair of DPs (i.e., one per party) is specific to a pre-determined set of functions $F$, a set of decryptors $D$, and an expiration time $t_e$.

**Assumptions.** A trusted process separately attests the PSs and signs their public keys, verifies that the DPs correctly authorize decryptors in $D$, and verifies that the specification of $F$ matches the source code of the DPs. In § 5, we describe how we implement this trusted process in CoVault in a dectralized fashion using community approval.

**System initialization.**

1) Each party instantiates its PS; then, each PS generates its party's private-public keypair.
2) Each party starts its DPs; then, each DP is remotely attested and provisioned by its PS with cryptographic keys, including the key to decrypt its share of data (in the sense of standard asymmetric cryptography).
3) Each PS holds information on the set of triples $[F, D, t_e]$. For each triple, this information consists of the specification of each function in $F$, the public keys of the authorized decryptors in $D$, the data expiration date $t_e$, and the public keys of both DPs that implement that $[F, D, t_e]$.
4) The PSs and DPs can be safely shut down and re-started from their sealed state [27] without requiring re-attestation.

### 4.2. Secret-sharing and encryption

An encryptor wishing to encrypt a cleartext data item $m$ first generates a single-use random value $r$. The shares of $m$ are then $r$ and $m \oplus r$, where $\oplus$ is bitwise exclusive-or (XOR). Encryptors can contact the PSs to retrieve information on the set of $[F, D, t_e]$ they can consent to, as well as the public keys of the DPs that implement them. If an encryptor consents to a particular $[F, D, t_e]$, they secret-share their data, encrypt each share with the appropriate public key, and send the encrypted share to the respective DP. Secret sharing ensures data confidentiality, while subsequently encrypting the shares prevents data misuse; in fact, only correctly attested TEEs (i.e., provisioned with the corresponding decryption keys) can decrypt the shares.

### 4.3. Garbled circuits in the malicious model

The DPs run 2PC using Yao's garbled circuits [80] (GCs) in the *malicious* threat model. Yao's original GC protocol

is secure only in the *semi-honest* threat model: One party (the *generator*) generates a garbled circuit representing the computation, and the other party (the *evaluator*) blindly evaluates the circuit, using oblivious transfer (OT) to ask the generator for the garbling of the evaluator's inputs. A malicious generator can easily generate an incorrect circuit and thereby compromise both data confidentiality and result integrity.

Extensions of Yao's protocol to the malicious setting are usually very expensive [77], but one recent extension – the DualEx protocol [48] – is reasonably efficient. DualEx runs *two* instances of a semi-honest GC protocol concurrently on parallel cores, with the roles of the generator and evaluator swapped. Both runs use a maliciously-secure OT protocol [79]. Each run reveals the computation result to the respective evaluator, so both parties learn the result. A final malicious-secure circuit [77] checks that the results match. The 1-bit result of this check is revealed to both parties. A malicious party can use this check to learn any one bit of the other party's input. However, if the result is false, the honest party refuses to communicate further with the other party, so this leak cannot be amplified by repetition. Hence, DualEx is maliciously secure *except* for a 1-bit leak in the information-theoretic sense. This 1-bit relaxation in confidentiality trades-off with the better runtime efficiency compared to other maliciously secure GC protocols, which is why we build on DualEx.

**Our extension to DualEx.** We design a modified DualEx protocol that does not reveal the output of the computation to the two parties, but instead to the decryptor, in order to match the needs of our FE design (strawmen S1–S3). Specifically, our *modified DualEx* reveals different *shares* of the output to the two parties. The two parties send these shares to the decryptor, who reconstructs the output. (Note that this is security-equivalent to, but more efficient than, encrypting the output with the decryptor's public key within 2PC, as in strawmen S1–S3.)

Our modified DualEx works as follows. Let $y$ be the actual computation output. The two parties first run standard DualEx to compute $y \oplus r_1 \oplus r_2$, where $y$ is the actual computation output, and $r_1$ and $r_2$ are random values respectively provided by each party as additional inputs to the circuit, and $\oplus$ is bitwise exclusive-or (XOR) (the XOR operation can be implemented very cheaply in garbled circuits [50]). The computed value $y \oplus r_1 \oplus r_2$ is returned to the two parties by the standard DualEx. Then, the parties recover shares of $y$ as follows: Party 1 computes the XOR of its random input $r_1$ and the output $y \oplus r_1 \oplus r_2$, while Party 2 ignores the output and uses $r_2$ as its share. These resulting values, namely $y \oplus r_2$ and $r_2$, are the shares of $y$.

For integrity, the shares $r_2$ and $y \oplus r_2$ are also MAC'd *within* 2PC, using the protocol in § 4.4. The two parties eventually reveal their shares and the MACs to the decryptor, who checks the MACs and reconstructs the output. We prove the security of our modified DualEx protocol in § E.3.

### 4.4. Secret-sharing with MACs

Following strawman S3, our design uses MACs over input and output shares for integrity. First, encryptors locally compute MACs on the shares of data, before encrypting and uploading them to the DPs; the DPs check the MACs on input shares within 2PC before computing any function. Second, the DPs compute MACs on the shares of the function's output in 2PC, as described in the previous subsection.

To this end, we extend the secret-sharing scheme of § 4.2 with MACs as follows: To securely share $x$, generate a random key $k$, compute $t \leftarrow \mathsf{HMAC}_k(x)$, secret-share $k$ into $k_1, k_2$ and $x$ into $x_1, x_2$ using the scheme of § 4.2. Output $(x_i, k_i, t)$ to party $i$ for $i = 1, 2$. The two parties can jointly verify the MAC later, but neither party can manipulate its share $x_i$ or $k_i$ without failing subsequent verification. Our implementation uses a SHA-3 HMAC. We formally prove the security of this construction in § E.2.

## 5. CoVault's design

CoVault uses the FE-like construction from § 4 as the core of a secure analytics platform. Next, we sketch how this is done.

**Roadmap.** A data source (encryptor) consents to having its data used for queries (functions) in $F$ by queriers in $D$ for a period ending at time $t_e$; we call the triple $[F, D, t_e]$ a *query class*. A query class is the unit of consent: Data sources decide on the query class(es) they wish to consent to, and encrypt their data accordingly. Once time $t_e$ has passed, the DPs that implement the associated class cease to decrypt/query the data, effectively making the data in the query class inaccessible. Authorized queriers can request a given query to be run on the data contributed to a query class by contacting the DPs implementing that query class. Next, we sketch how operations in CoVault are mapped to the operations of the FE-like construction defined in § 3.1.

**Defining a query class.** To define a new query class, an interested analyst calls the $Setup$ operation of both PSs, passing the appropriate $F, D, t_e$ as arguments. The PSs update their configuration state accordingly and publish the vector of public keys $MPK[F, D, t_e]$ and the public keys of the DPs associated with the new query class.

**Contributing data.** Data sources choose a query class to which they wish to contribute their data. Then, they perform the $Enc$ operation locally using the vector of public keys associated with the chosen query class to produce the encrypted and MAC'd shares of their data, which they then send to the DPs associated with the query class.

**Querying the database.** To run a query $f$, a querier connects via secure channels to the two DPs that implement the appropriate query class. The querier first performs a $KeyGen$ operation on both DPs to obtain $SK_f$ and then performs a $Dec$ operation on both DPs to obtain the shares of the desired query result. These two operations check that the function and the decryptor are in the query class, and that the expiration time of the query class has not passed.

**Community approval process.** CoVault relies on a community approval process to justify trust in its implementation. In principle, each data source could inspect the source code of the PSs, the DPs of relevant query classes, as well as the build chain used to compile the system, in order to verify that all components are correctly implemented according to their specification, and then remotely attest the PSs—which form the system's technical root of trust—to make sure their initial measurement hash is as expected. However, this approach is impractical, because most data sources lack the technical expertise and resources to perform these actions. The community approval process provides a level of indirection. Interested community experts inspect the system components and publish signed statements that a PS or DP with a given measurement hash correctly implements a specification. Separately, community members may remotely attest the PSs and sign their public keys if their measurement hashes are as expected. Data sources can rely on the assessments of community experts they trust. We describe community approval in more detail in § A.

**Privacy from authorized queriers.** To ensure that authorized queriers cannot infer private data from the results of authorized queries, query implementations should provide strong privacy guarantees, ideally differential privacy. Queries with weak privacy should not be approved by the community experts. Auxiliary state needed for enforcing privacy, such as residual privacy budgets in the case of differential privacy, can be stored in CoVault's database itself (secret-shared, MAC'd and encrypted like other data), and the enforcement of the privacy policy, such as updates to residual privacy budgets, can be done in 2PC. The DPs can also maintain a log of executed queries for later audit.

### 5.1. Security Analysis and Proofs

Our design attains the confidentiality and integrity properties in our threat model (§ 3.2), which assumes that at least $n - t$ of the TEEs in which DPs run are uncompromised and free of side-channel exploits. This assumption, together with our community approval process, implies that at least $n - t$ of the DPs to which a data source provides shares are completely honest, i.e., they follow prescribed protocols and do not leak additional information, even under active attacks (side-channel or others).

Hence, the security of CoVault's design reduces to proving that the MPC scheme we use is actually $t$-of-$n$ secure in the malicious model. Our design uses $n = 2$, $t = 1$, and relies on a combination of our modified DualEx protocol for MPC (§ 4.3), and our secret-sharing with MACs (§ 4.4). We formally define and prove the security of these constructions and the end-to-end security of CoVault's design in § E.

## 6. Data processing

Each of the two parties has access to its own data store, which contains that party's shares. For simplicity, we refer to both data stores as CoVault's db (database), but we note that any fetch operation on this db consists of two independent, parallel fetches, one per party. In this section, we discuss the general structure of CoVault's db and how we run queries on large datasets *scalably*. Due to space constraints, we defer a description of data ingress processing to § B.

### 6.1. Database

CoVault's (encrypted, MAC'd) data shares can be stored in any untrusted database. In general, the db access pattern during a query can leak secrets. A general solution is ORAM [41]; however, ORAM – even a read-only variant – is expensive and does not naturally permit concurrent access. Therefore, CoVault relies on the private information retrieval (PIR) scheme described in the next paragraph. Queries that do not access rows data-dependently do not use this scheme and their db accesses can be optimized (§ 6.3).

**Oblivious database access.** CoVault uses a simple random shuffling scheme to shuffle tables that are accessed with non-public keys. The shuffling hides which records are accessed, so this forms a simple PIR scheme with no query-time overhead. Shuffling is implemented by a pair of DPs, $DP_1$ and $DP_2$, without 2PC. $DP_1$ locally shuffles the rows of a table using a randomly generated permutation $\rho_1$ that it keeps secret. The shuffling is implemented obliviously but locally on $DP_1$; whenever two rows are read to potentially be swapped, they are re-encrypted within $DP_1$, so an outside observer cannot determine whether they were actually swapped. $DP_1$ then passes the shuffled table to $DP_2$, which re-shuffles once more with a permutation $\rho_2$, that it ($DP_2$) generates uniformly at random and keeps secret. The resulting overall permutation $\rho_1 \circ \rho_2$ remains unknown to either DP alone and is uniformly random as long as one DP is honest, so this scheme is robust to one malicious party per our threat model. During query execution, the DPs provide $\rho_1$ and $\rho_2$ as private inputs to 2PC. Using this information, $\rho_1 \circ \rho_2$ is computed *within* 2PC, and the indices of rows in the end-to-end shuffle are calculated.

Shuffling cannot hide whether two lookups access the same row. Therefore, to avoid frequency attacks, a shuffled table is used for one query only, and a query never fetches the same row twice. Reshuffling can be done ahead of time, so that fresh shuffled tables are readily available to queries.

### 6.2. Query processing

The unit of querying in CoVault is a SQL filter-groupby-aggregate (FGA) query, which we call a *basic* query. In general, a querier may make a series of *data-dependent* basic queries with query parameters of later queries depending on the results of earlier queries. In the following, we first describe how CoVault executes basic queries, and then how it handles data-dependent series of basic queries.

**FGA or basic queries.** A FGA/basic query has the form:
SELECT `aggregate`([DISTINCT] `column1`), ...
FROM `T` WHERE `condition` GROUP BY `column2`
Here, `aggregate` is an aggregation operator like SUM or

COUNT. The query can be executed as follows: (i) filter (select) from table `T` the rows that satisfy `condition`, (ii) group the selected rows by `column2`, and (iii) compute the required `aggregate` in each group. A straightforward way of implementing a FGA query is to build a *single* garbled circuit that takes as input the two shares of the entire table `T` and implements steps (i)—(iii). However, this approach does not take advantage of *core parallelism* to reduce query latency. Moreover, the size of this circuit grows super-linearly with the size of `T` and the circuit may become too large to fit in the memory available on any machine. To exploit core parallelism and to make circuit size manageable, CoVault relies on the observation that FGA queries can be implemented using map-reduce [33]. We first explain how this works in general (without 2PC) and then explain how CoVault does this in 2PC.

**Background: FGA queries with map-reduce.** Suppose we have $m$ available cores. The records of table `T` are split evenly among the $m$ cores.

Step (i): Each core splits its allocated records into more manageable *chunks* and applies a **map** operation to each chunk; this operation linearly scans the chunk and filters only the records that satisfy the WHERE `condition`.

Steps (ii) and (iii): These steps are implemented using a tree-shaped **reduce** phase. The 1st stage of this phase uses half the number of reducers as the map phase. Each **1st-stage reducer** consumes the (filtered) records output by two mappers, sorts the records by the grouping column `column2`, and then performs a linear scan to compute an aggregate for each value of `column2`. The output is a sorted list of `column2` values with corresponding aggregates. Each subsequent stage of reduce uses half the number of reducers of the previous stage: Every **subsequent-stage reducer** merges the sorted lists output by two previous-stage reducers adding their corresponding aggregates and producing another sorted list. The last stage, which is a single reducer, produces a single list of `column2` values with their aggregates.

**CoVault: FGA queries with map-reduce in 2PC.**

CoVault executes FGA queries by implementing the mappers and reducers described above as *separate* garbled circuits and evaluating the circuits in 2PC. Thus, CoVault inherits scaling with cores from the map-reduce paradigm.

The *challenge* here is that circuits can implement only a limited class of algorithms. In particular, a circuit is data-oblivious – it lacks control flow – and the length of the output of a circuit cannot depend on its inputs. However, common algorithms for sorting rely on control flow (they branch based on the result of integer comparison), and standard algorithms for filtering and merging lists produce outputs whose lengths are dependent on the values in the input lists. Hence, to implement mappers and reducers in circuits, we have to use specific algorithms that are data oblivious and pad output to a size that is independent of the inputs (this padded output size, denoted $d$ below, is an additional parameter of the algorithm; it should be an upper bound on the possible output sizes). In the following, we explain basic data-oblivious algorithms that CoVault relies on, and then explain how it combines them with padding when needed to implement mappers and reducers in circuits.

CoVault relies on the following standard data-oblivious algorithms implemented in circuits:

- A *linear scan* passes once over a list performing some operation (e.g., marking) on each element, or computing a running total.
- *Oblivious sort* on a list. While oblivious sort has a theoretical complexity $O(n\log(n))$, all practical algorithms are in $O(n(\log(n))^2)$. CoVault uses bitonic sort [15].
- *Oblivious sorted merge* merges two sorted lists into a longer sorted list. It retains duplicates. CoVault uses bitonic merge, which is in $O(n\log(n))$ [15].
- *Oblivious compact* moves marked records to the end of a list, compacting the remaining records at the beginning of the list in order. For this, CoVault uses an $O(n\log(n))$ butterfly circuit algorithm [42, Section 3].

CoVault uses these algorithms to implement mappers and reducers as follows. A CoVault **mapper** uses a *linear scan* to only mark records that do *not* satisfy the WHERE `condition` with a discard bit; it does not actually drop them, else the size of the output list might leak secrets. A **1st-stage reducer** *obliviously sorts* the outputs of two mappers, ordering first by the discard bit, and then by the GROUP BY criterion, `column2`. This pushes all records marked by the mappers to the end of the list, and groups the rest in sorted order of `column2`. Records with the same value of `column2` belong to the same group, so the reducer must consider them just once when computing the aggregate. To this end, it performs a *linear scan* to (a) compute a running aggregate for each unique group, and (b) mark all but one record in each group to be discarded. Finally, it does an *oblivious compact* to push all marked records to the end of the list. The unmarked records contain *unique*, sorted values of `column2` paired with corresponding aggregates. This output is truncated or padded to a fixed length $d$, which, as mentioned earlier, is an additional query parameter that should be an upper bound on the possible number of unique groups in `column2`. **Subsequent-stage reducers** are similar to the 1st-stage reducers, except that they receive two already sorted lists as input, so they use *oblivious sorted merge* instead of the more expensive oblivious sort.

The theoretical complexities of a mapper, 1st-stage reducer, and subsequent-stage reducer are $O(c)$ where $c$ is the chunk size, $O(c(\log(c))^2)$ and $O(d\log(d))$, respectively. The chunk size $c$ has a non-trivial effect on query latency: Larger chunks result in more expensive mappers and 1st-stage reducers, but fewer total number of mappers and reducers. In practice, we determine the chunk size empirically to minimize query latency; our experiments typically use $c = 10,000$ records.

**Running a series of data-dependent FGA queries.** If the data rows accessed by a FGA query *depend* on the output of an earlier query, information about database contents may leak via the db access pattern. To avoid such leaks, a data-dependent series of FGA queries uses previously shuffled

tables only (§ 6.1). Also, the number of records read must not depend on previous query results; to this end, 2PC adds dummy record requests.

### 6.3. Optimizations

**Indexing along public attributes.** If a table contains a non-sensitive, public attribute, we can speed up queries that filter on this attribute by creating *public indexes* on this attribute and fetching only filtered data, thus not needing to filter within 2PC. In this case, CoVault does not MAC each record individually, but computes MACs (one per column) on all records with the same public attribute value, as they would be fetched together. Of course, each data source must consent for a given attribute to be public: CoVault's public configuration contains the public attributes for each query class. In § 7.1, we show examples of tables (views) and queries that exploit such indexes on public attributes.

**Joins.** Join operations are very expensive when done obliviously [82], and even more expensive in 2PC. For instance, consider a self-join: each row of a table is compared to every other row of the same table. To hide access patterns, an oblivious self-join scans the whole table for each row in the table. However, if the access pattern reveals *only* public information, we can speed up joins. In fact, the access pattern is *not* sensitive if the data is joined on a public attribute. In these cases, CoVault performs joins outside 2PC. We describe an example of this optimization in § C.

**Materialized views.** A materialized view is a stored query result, maintained to speed up future queries that contain the same query as a subquery. CoVault supports materialized views, including those on pre-joined data. If a public attribute exists, CoVault stores materialized views with a public primary index. If no public attribute exists, or if a query requires secret-dependent data access to a view, CoVault stores shuffled copies of the view, suitable for PIR (§ 6.1).

**Keeping data in garbled form.** Data passed between data sources and DPs, between pairs of DPs along the map-reduce pipeline, and between DPs and queriers must be MAC'd to prevent either party from modifying its shares. MAC computation in garbled circuits is expensive to the point that it can be the dominant cost of 2PC (see § C.3). While the use of MACs to protect data between data sources and DPs, and between DPs and queriers is unavoidable, we can eliminate MACs between DPs along the map-reduce pipeline by passing values between DPs of consecutive stages in garbled form (i.e., their in-circuit representation). This eliminates the need for the MACs within map-reduce. We use this optimization in our prototype implementation.

## 7. Evaluation

We implemented CoVault using EMP-toolkit [76], a library that compiles and executes C/C++ programs in garbled circuits-based 2PC, using recent optimizations such as Half Gates [81] and FreeXOR [50]. We use bitonic sort and bitonic merge from the EMP library, and implement our own primitives for compaction, linear scan, and the MAC of §4.4 (based on a SHA-3 HMAC, using a pre-existing circuit for the core SHA-3 loop). We compose these to implement microbenchmarks and queries. Redis (v5.0.3, non-persistent mode) is used as the db, and we use the optimizations mentioned in § 6.3. We implemented the two-party shuffle operation from § 6.1 and we hold shuffled views in memory on a DP (rather than in Redis) for efficient re-shuffling.

We run 2PC between two parties on Intel and AMD TEE CPUs, respectively. We use 7 machines with Intel®Xeon®Gold 6244 CPU 3.60GHz 16-core processors (2 sockets, 8 cores/socket, 1 thread/core) and 495GB RAM each, and 7 machines with AMD EPYC 7543 2.8GHz 32-core processors (2 sockets, 16 cores/socket, 1 thread/core) and 525 GB RAM each. All machines run Debian 10 and are connected via two 1/10GB Broadcom NICs to a Cisco Nexus 7000 series switch. In addition, each pair of Intel and AMD machines is directly connected via two 25Gbps links over Mellanox NICs; these are used for 2PC. Frequency scaling is disabled on the Intel machines, where the CPU governor is set to performance mode; this feature is not available on the AMD machines, which only have the nominal frequency option.

On both types of machines, we run computations on Linux Debian 10, hosted inside VMs managed using libvirt 5.0 libraries, the QEMU API v5.0.0, and the QEMU hypervisor v3.1.0. On the AMD machines we run the VMs in SEV-SNP TEEs provided in the sev-snp-devel branch of the AMDSEV repo [1], along with their patched Qemu fork. On the Intel machines, we run the VMs without a TEE, because CPUs with TDX are not yet available and SGX has very stringent memory limits. We believe the resulting performance of 2PC is nevertheless representative, because it is limited by the slower AMD CPUs, which do use TEEs. We use at most 16 cores in the AMD SNP-SEV VMs to match the cores available on the Intel machines (as cores are used in pairs).

### 7.1. Example scenario: Epidemic analytics

We evaluate CoVault in the context of epidemic analytics. We note that epidemic analytics is only an example scenario: CoVault's query processing overhead is independent of the data semantics, which impacts at the most the choice of the upper bound on the possible output sizes (see § 6.2). Consequently, this evaluation applies to any other application scenario implementing queries with the same sequence of FGA operations and similar input and output sizes.

We use synthetic data representing time series of GPS locations and pairwise Bluetooth radio encounters reported by smartphones [54], [71]. Real data of this sort would be sensitive, and we assume that users would provide such data for the purpose of privacy-preserving epidemic analytics only to a secure platform like CoVault.

Figure 3: Schemas of two materialized views used for epidemic analytics. The first column in *italics* is a public index, while the rest of the record is confidential. (The full schemas are shown in Figure 7).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $T_E$ | *space-time-region* | eid | did1 | did2 | ... | | | |
| $T_P$ | *time-epoch* | did1, time | did2 | duration | prev | next | time | ... |

Figure 4: Queries used in the evaluation. The selections on the public attributes `space-time-region` and `time-epoch` are done outside 2PC using the public indexes of $T_E$ and $T_P$. `R` is a set of space-time-regions.

**(q1)** Histogram of #encounters, in space-time regions `R`, of devices in set `A`

```
SELECT HISTO(COUNT(*)) FROM T_E
WHERE did1∈A AND space-time-region∈R
```

**(q2)** Histogram of #unique devices met, in space-time regions `R`, by each device in set `A`

```
SELECT HISTO(COUNT(DISTINCT(did2))) FROM T_E
WHERE did1∈A AND space-time-region∈R
```

**(q3)** Count #devices in set `B` that encountered a device in set `A` in the time interval `[start,end]`

```
WITH TT AS
 (SELECT * FROM T_P
  WHERE start<time-epoch<end)
SELECT COUNT(DISTINCT(did2)) FROM TT
WHERE did1∈A AND did2∈B
      AND start<time<end
```

We detail the data upload process and CoVault's data pre-processing steps in § C. The pre-processing results in two materialized views $T_E$ and $T_P$, whose schemas are shown in Figure 3. Each view has a *public*, coarse-grained index, which is shown in *italics* font. $T_E$ is a list of pairwise encounters with an encounter id (eid), anonymous ids of the two devices (did1, did2) and additional information about the encounter (indicated by dots ... in the table). Its public index is a coarse-grained *space-time-region*, which generally is of the order of several km$^2$h. Records within the same space-time-region are stored together in CoVault's db. The resolution of space-time regions depends on public information and is chosen so as to achieve an approximately even number of records per space-time region. The exact number of records in each space-time region is obfuscated by padding all regions to the same length (§ C). The second view $T_P$ contains encounters privately indexed by individual device ids and the times of the encounter reports (did1, time). A record also contains pointers to the previous and next encounter of the same device. These pointers are used to traverse the timeline of a given device. The public index is a coarse-grained interval of time (∼1h) in which the encounter occurred. We call this interval a *time-epoch*. $T_P$ is used by data-dependent queries as explained later, so this view is randomly shuffled (§ 6.1).

Our evaluation uses the three queries q1–q3 shown in Figure 4, which we developed in consultation with an epidemiologist. Queries q1 and q2 are histogram queries on frequencies of encounters. They bin devices in a given set `A` by the number of total encounters and the number of unique encounters with distinct devices they had in a given, arbitrary space-time region `R`. Such queries can be used to understand the impact of contact restrictions (such as closure of large events) on the frequency of contacts between people. These queries run on $T_E$. Query q3 asks how many devices from set `B` encountered a device from the set `A` within a given time interval. This query can be used to determine if two outbreaks of an epidemic (corresponding to the sets of devices `A` and `B`) are directly connected. A related query, discussed in §C.4, extends q3 to indirect encounters between `A` and `B` via a third device.

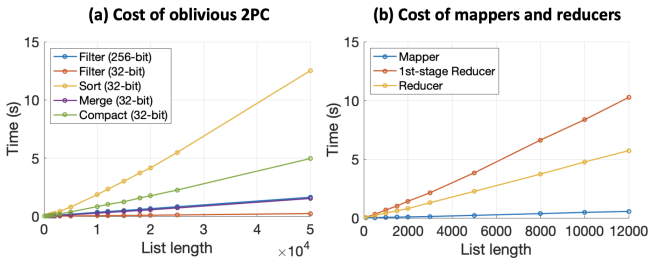



Figure 5: Processing time vs. workload size

## 7.2. Microbenchmarks

We first report the costs of basic 2PC primitives (§ 6.2), individual mappers and reducers (§ 6.2), and random shuffling (§ 6.1). All experiments reported here are on a single pair of cores, one Intel and one AMD. We measure the cost of only one of the two executions of DualEx since the two executions are completely independent everywhere in CoVault, execute in parallel, and synchronize only once at the end of every query for DualEx's equality check (which is performed only at the last reducer stage). The reported experiments ran the 2PC generator on an Intel core and the evaluator on an AMD core, but we have tried the opposite configuration and the results are similar.

**Primitives.** Figure 5 shows the time taken to execute the basic oblivious 2PC primitives from § 6.2—linear scans on 32-bit and 256-bit records, sort, sorted merge, and compact on 32-bit records—as a function of the number of records. Each reported number is an average of 100 measurements. The coefficient of variation for input size 100 elements is 17% on average and at least 8% for each operation. For other input sizes, it is below 4% on average and at most 5% across all operations.

The trends are as expected: The cost of linear scans grows linearly in the input size, while the costs of sort, sorted merge and compact are slightly super-linear. In particular, the cost of sorting is significantly higher than that of compaction, which is why our reduce trees sort only in the first stage and then use compact only (§ 6.2).

**Map and reduce.** The basic units of query processing are map and reduce operations. Figure 5 shows the average

time taken to execute specific but representative map, 1st-stage reduce, and subsequent-stage reduce operations as a function of the number of input records on a single core pair. The specific operations are from query q2 in Figure 4. The map operation takes records from $T_E$, pre-filtered to a given space-time region. It linearly scans the records to mark those whose did1 matches a given device id and projects every marked record to just a 32-bit fingerprint of did2, in effect finding all devices (did2s) that met the given device id. The 1st-stage reduce takes the outputs of two maps, sorts and merges them, marks duplicate fingerprints for deletion, and compacts, in effect *counting* the number of unique devices. The subsequent-stage reduce does the same, but on the output of the 1st-stage reduce, so it does not have to sort. The results are as expected: map is linear in the number of input records, while reduce is slightly super-linear.

**Shuffling.** Next, we evaluate the cost of random shuffling to produce shuffled encounters views (§ 6.1). First, random shuffling requires a *one-time* pre-processing step to MAC and encrypt every record separately. A basic unit (2 AMD + 2 Intel cores) in our setup pre-processes records of schema $T_P$ in 17.8ms per record, with standard deviation $<$ 0.9ms (6%). A country with population 80M produces 11.85B encounters/day, assuming conservatively 100 encounters/person/day in rural areas and 200 encounters/person/day in urban areas; pre-processing all these encounters in a day requires 2,441 basic units, or 4,882 core pairs. An urban town with a population of 200K would produce 40M encounters/day and similarly require 8 basic units, or 16 core pairs.

Each shuffle itself requires two *sequential* oblivious sorts on CPUs of different types (outside 2PC). Shuffling a view of 617M records – a conservative upper bound on the number of encounters generated in 1h in a urban city of 3M people – takes about 56min in our setup. Shuffling one-tenth the number, 61M records, takes 4 minutes. Variances are negligible. The scaling is super-linear because oblivious sort (even outside 2PC) runs in $O(n(\log(n))^2)$ steps.

Queries use a shuffle only once, but shuffles can be generated ahead of time in parallel. Because sorting takes place on one machine at a time, a single pair of cores can, in less than 1h, produce 2 different shuffles for encounters generated in 1h. Therefore, we can prepare shuffles for $q$ queries using $q/2$ pairs of cores continuously.

**Bandwidth.** 2PC streams large garbled circuits from the generator to the evaluator, and the generator also transmits the garbling of the evaluator's inputs. During a series of sort and scan operations, the average bandwidth from the generator to the evaluator, measured with NetHogs [2], is ∼2.72Gbps. The bandwidth from the evaluator to the generator is negligible in comparison (0.11Gbps). With DualEx, the average bandwidth would be 2.72Gbps in each direction. Thus, 16 active cores on each machine need a bandwidth of 22.64Gbps in each direction, which our 2x25Gbps links can easily support.

**Storage overhead.** When storing databases in non-garbled form (i.e., without the optimization of § 6.3), CoVault imposes slightly more than 2x storage overhead on the db. The 2x comes from the need to store two shares per value, and there is slightly more storage for MACs depending on the size of the space-time region or time-epoch (one MAC per column per region or time-epoch). When storing in garbled form, as we actually do, the overhead is 512x: Each bit of data expands to 128 bits due to garbling, which doubles due to secret sharing, and further doubles due to the two parallel garbled circuits of DualEx. Based on the conservative calculation above, 11.85B encounters/day amounts to at most 2.40 TB (non-garbled) and 618.78 TB (garbled) of added storage per day for our largest view $T_E$.

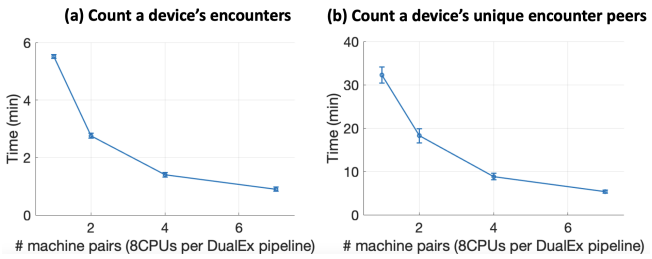


Figure 6: Query latency vs. available core pairs for the basic query of queries (q1) and (q2) from Figure 4.

## 7.3. Query latency

We empirically evaluate end-to-end query latency, a key performance metric, for the queries q1–q3 of Figure 4.

**Queries q1 and q2.** Queries q1 and q2 run on the view $T_E$. The queries fetch only the records that are in the space-time-region `R`, using the public index on $T_E$. This significantly reduces the size of the input table and the query latency. In both queries, we iterate over devices in the given set `A`. For each device $a$ in `A` we issue a basic FGA query. For q1, this basic query counts the number of encounters of device $a$ in `T`. For q2, this basic query counts the number of unique devices that device $a$ met. Note that basic queries for all devices in `A` can be done in parallel, independently. We empirically evaluate the latency-resource trade-off for the basic queries of q1 and q2 by varying the number of available machine pairs from 1 to 7. The experiments here are end-to-end and perform both runs of DualEx (in parallel, on two pairs of cores).

Figure 6 shows the latency of q1's basic query against the number of available machines. All points are averages of 10 runs (std. dev. $<$ 4.7%). As input, we use a table with 28M encounter records (corresponding to a conservative upper bound on encounters generated in a space-time region with 10k data sources reporting 200 encounters/day over 14 days). We process these records in chunks of size 10k, which minimizes the latency empirically. Mappers filter input encounters to those involving the specific user, and reducers just aggregate the number of encounters. The latency is almost inversely proportional to the number of core pairs available, since the bulk of the work is map-reduce locally

on each machine (cross-machine reduce is done only once at the end).

Figure 6 shows the latency of q2's basic query in the same setup. The map phase is unchanged, but each reduce now combines and removes duplicates from two lists of encountered devices. As explained in § 6.2, we need to specify an upper bound on the size of each reducer's output. Here, we fix this to 500 (i.e., we assume that a person will encounter at most 500 unique devices in 14 days). Numbers are averages of 10 runs each, and coefficients of variation are below 9.1%. Again, the query latency varies almost inversely with the number of machines, but the costs are higher than those of q1 since reducers do more work.

**Scaling.** From measurements of individual mappers and reducers in queries, we can extrapolate the number of machines needed to attain a certain query latency with a given number of input records. For example, if the input was 10 times larger (280M records), answering q2's basic query in 10min needs 736 core pairs (i.e., 46 8-CPU machines). The details of our extrapolation algorithm are explained in § D .

**Query q3.** This query asks for the number of devices in `B` that directly encountered a device in `A` in the time interval `[start,end]`. To implement q3, we traverse the trajectories of devices in `B` backwards in time, by iterating over *time-epochs*, from the time-epoch that contains `end` to the time-epoch that contains `start`. This iteration over time-epochs is done outside 2PC since time-epochs are public. The data for each time-epoch is successively loaded into a temporary table, called `TT` in the query, and this table is then processed via a query in 2PC. The 2PC query traverses the trajectory of each device in `B` from the device's last encounter in the time-epoch to their earliest encounter, by following encounter pointers. For this traversal, 2PC makes data-dependent queries to the time-epoch's records in `TT` but since the records in each time-epoch are randomly shuffled per our PIR scheme, this does not reveal any secrets. To ensure that no secrets leak via the *number* of lookups in an time-epoch, we use a fixed, conservative number of lookups per device (in `B`) per time-epoch, fetching dummies when needed (the size of `B` is part of the query, hence, public). Every other device encountered by a device (in `B`) during the traversal is checked for membership in `A`, via a Bloom filter initialized with devices in `A` (the Bloom filter is implemented in 2PC). Whenever the membership test succeeds, we mark the device in `B` as having encountered someone in `A`. At the end, we simply count the number of marked devices.

In our experiments, it took (in 2PC) a constant 52ms to initialize the Bloom filter for $A = 10$, 27ms (std. dev. 2ms) to fetch an encounter from a shuffled view, 43ms (std. dev. 5ms) to check set membership against the Bloom filter, and 10ms (std. dev. 2ms) to check that an encounter's time is between `start` and `end`. Assuming $B = 10$, a total of $e = 336$ time-epochs (corresponding to a period of 14 days with 1h time-epochs), and at most $n = 30$ encounters/person/h, the total query latency comes to $52+(27+10+43)\cdot e\cdot n\cdot|\mathtt{B}|$ = 2.24 hours.

Note that our PIR scheme improves query latency significantly by by enabling data-dependent accesses. If data-dependent accesses were not possible, we would have to scan all encounters of *all* encounters in the time interval `[start,end]`. Assuming, as before, that 11.85B encounters are generated in a country every day, and the interval `[start,end]` is 14 days long, this data-*in*dependent approach would have to process nearly $11.85\text{B}\cdot 14 = 165.9\text{B}$ encounter records in 2PC. By contrast, the data-dependent approach above processes a total of $336 \cdot 30 \cdot 10 = 100,800$ records in 2PC, which is a saving of over 6 orders of magnitude in the number of records fetched from the db and processed in 2PC.

**Summary** We conclude our evaluation with a list of key takeaways.

First, CoVault's combination of garbled circuits and map-reduce scales well: CoVault can analyze large datasets with reasonable latency using proportionally more cores.

Second, the core requirements are moderate even for country-scale datasets if query latencies of the order of hours are acceptable. For example, for a mid-sized country with 80M people, we estimate that continuously ingesting all incoming encounter records (11.85B records/day) requires 1,660 core pairs on a continuous basis (see § C.3), shuffling incoming records continuously – if needed for data-dependent queries – needs another 4,882 core pairs on a continuous basis (§ 7.2), and running an epidemic analytics query like q2 on 14 days of such records (165.9B records) within 10 hours requires an additional 5,600 core pairs engaged for the 10 hours of the query's execution. Although high in absolute number, these core requirements are at a level that is already available in the data centers of individual research institutes today, and we believe that such an investment is justifiable when the analytics has a sufficiently high socio-economic benefit. The core requirements also fall directly with the size of the datasets. For example, for a urban city with 200k people recording 200 encounters/day each (a total of 40M encounters/day), the core requirements for continuous data ingestion, continuous shuffling and answering q2 on 14 days of records within 10 hours would be 6, 16 and 20 core pairs, respectively.

## 8. Related work

We discussed work on analytics platforms secured by TEEs only in § 1. Here, we discuss other related work.

**Data-oblivious analytics.** Data analytics can be implemented using data-oblivious algorithms to prevent information leaks through memory access patterns. Cipherbase's oblivious extension [12] proposes optimized, oblivious implementations of query operators but, to the best of our knowledge, this extension was not implemented. Ohrimenko, Schuster et al. [58], Ohrimenko, Costa et al. [57], and $M^2R$ [34] use similar techniques to mitigate side channels in map-reduce and machine-learning queries. Opaque [82], OCQ [32], and StealthDB [43] are secure analytics platforms that run encapsulated in a (single) TEE but, additionally, implement query operators using oblivious

algorithms. ObliDB [38] extends Opaque and Cipherbase with optimized operators that do not need to scan the whole table for every query. Although CoVault also relies on oblivious algorithms, its threat model is stronger: it does *not* rely on the honesty of any single party or TEE implementation.

ORAM [41], [75], [35] is a general technique for hiding the memory access pattern of any computation. However, ORAM is costly compared to custom oblivious query operations. For read-only queries, private information retrieval (PIR) offers a more efficient alternative [28], [31], [59], [74]. CoVault avoids the costs of ORAM and PIR for queries whose access patterns reveal no secrets. For other queries, CoVault uses a new PIR scheme based on ahead-of-time random shuffling (§ 6.1).

**Analytics using secret sharing.** Like CoVault, Obscure [45], SMCQL [16], Crypt$\epsilon$ [26] and GraphSC [55] use secret sharing to distribute trust over multiple parties, and MPC or partially homomorphic encryption to process queries. However, these systems rely on an undischarged assumption about the non-collusion of the parties. CoVault additionally encapsulates the parties in TEEs of different types to reduce this assumption to the security of $n-t$ TEE types.

**Encrypted databases.** Several systems use variants of homomorphic encryption (HE) to compute aggregates on encrypted data [26], [63], [23], [60], [22]. HE's confidentiality property relies only on cryptographic assumptions about the adversary's power. This threat model is *seemingly* stronger than CoVault's but, in an analytics platform, decrypting query results after HE operations requires a trusted party or MPC [65], which weakens the actual threat model to something similar to our strawman S1 (or weaker). Additionally, full HE is very expensive [72], while partial HE restricts supported queries. Furthermore, none of the systems mentioned above protect against side-channel leaks, which CoVault does. Early encrypted databases like CryptDB [63] used even weaker forms of encryption like deterministic encryption or order-preserving encryption, which are insecure in practice [44], [19]. Blind Seer [61] uses strong encryption combined with 2PC to traverse a specialized index, but leaks information by exposing its search tree traversal and supports only a limited set of search queries.

**Federated analytics.** Federated analytics (FA) systems do not centralize data; queries requires a joint computation between all data sources. The absence of data centralization alleviates many confidentiality concerns faced by centralized databases like CoVault, but all federated systems face scalability issues: a large percentage of data sources must be online at any time, and data sources have high bandwidth and computational costs for query processing (unlike data center servers, data sources may be very resource constrained). Additionally, if done naively, query processing leaks information among data sources and to queriers. Security- and privacy-conscious FA systems like ShrinkWrap [17], Honeycrisp [66], Orchard [67], and Mycelium [65] deploy techniques like differential privacy and HE to aggregate data securely. However, these techniques are expensive and often limit the expressiveness of queries.

By way of example, with the exception of Mycelium, no secure FA system we know of supports graph queries, such as query q3 from § 7.1 or q4 from § C.4. Even Mycelium's protocol supports only those graph queries where a limited-hop local neighborhood of each vertex needs to be explored. Mycelium relies on level-homomorphic encryption and zero-knowledge proofs for confidentiality- and integrity-preserving data aggregation. The corresponding protocols are executed on data sources, requiring the data sources to contribute substantial computational power to query processing. CoVault also uses expensive 2PC, but this expensive computation is limited to a data center; data sources only use simple XOR operations, standard hybrid encryption, and standard MACs to create data shares. Finally, CoVault can filter data on *public* attributes without expensive operations, using public indexes (§ 6.3). In secure FA, all data sources must participate in all queries even if their data is not selected, leading to needless resource consumption. In summary, CoVault and secure FA operate at very different points in the threat model-efficiency-expressiveness space.

## 9. Conclusion

CoVault is a secure anaytics platform based on a new FE-like construction, which is secure under a strong threat model that tolerates compromise and side-channel attacks against any one TEE implementation. Experimental results show that CoVault can perform powerful analytics, while scaling out to large databases using map-reduce parallelism.

CoVault's efficiency can be further improved using techniques that we plan to explore in future work: (i) opportunistic use of partially-homomorphic secret sharing to compute certain operations locally without MPC, (ii) use of a combination of arithmetic and boolean circuits for MPC, and (iii) use of more than 2 parties, since MPC over 3 or more parties can be significantly cheaper than 2PC [25], [10] (this requires the availability of more than 2 TEE types, which will likely be the case in the future).

## References


[1] GitHub: AMDSEV repo. https://github.com/AMDESE/AMDSEV/tree/sev-snp-devel. Accessed: 2021-12-10.

[2] GitHub: Nethogs repo. https://github.com/raboof/nethogs. Accessed: 2021-12-24.

[3] "How much energy do data centers really use?", 2020. https://energyinnovation.org/2020/03/17/how-much-energy-do-data-centers-really-use/.

[4] Council approves data governance act, 2022. https://www.consilium.europa.eu/en/press/press-releases/2022/05/16/le-conseil-approuve-l-acte-sur-la-gouvernance-des-donnees/#:~:text=The%20Data%20Governance%20Act%20(DGA,data%20altruism%20across%20the%20EU.

[5] Michel Abdalla, Florian Bourse, Hugo Marival, David Pointcheval, Azam Soleimanian, and Hendrik Waldner. Multi-client inner-product functional encryption in the random-oracle model. In *International Conference on Security and Cryptography for Networks*, pages 525–545. Springer, 2020.

[6] Bethania de Araujo Almeida, Danilo Doneda, Maria Yury Ichihara, Manoel Barral-Netto, Gustavo Correa Matta, Elaine Teixeira Rabello, Fabio Castro Gouveia, and Mauricio Barreto. Personal data usage and privacy considerations in the covid-19 global pandemic. *Ciencia & saude coletiva*, 25:2487–2492, 2020.

[7] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. White paper at https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020. Accessed: 2020-05-27.

[8] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *Annual Cryptology Conference*, pages 657–677. Springer, 2015.

[9] Apple and Google. Privacy-Preserving Contact Tracing. https://www.apple.com/covid19/contacttracing. Accessed: 2021-12-14.

[10] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 843–862. IEEE Computer Society, 2017.

[11] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramarathnam Venkatesan. Orthogonal Security With Cipherbase. In *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*, January 2013.

[12] Arvind Arasu and Raghav Kaushik. Oblivious Query Processing. *CoRR*, abs/1312.4012, 2013.

[13] Sumeet Bajaj and Radu Sion. TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 205–216, New York, NY, USA, 2011. Association for Computing Machinery.

[14] Gilles Barthe, Roberta De Viti, Peter Druschel, Deepak Garg, Manuel Gomez-Rodriguez, Pierfrancesco Ingo, Matthew Lentz, Aastha Mehta, and Bernhard Schölkopf. PanCast: Listening to Bluetooth Beacons for Epidemic Risk Mitigation. *arXiv preprint arXiv:2011.08069*, 2020.

[15] K. E. Batcher. Sorting Networks and Their Applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS 1968 (Spring), pages 307–314, New York, NY, USA, 1968. Association for Computing Machinery.

[16] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. SMCQL: Secure Querying for Federated Databases. *Proceedings of the VLDB Endowment*, 10(6):673–684, February 2017.

[17] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. ShrinkWrap: Efficient SQL Query Processing in Differentially Private Data Federations. *Proc. VLDB Endow.*, 12(3):307–320, 2018.

[18] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.*, 33(3), August 2015.

[19] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. The Tao of Inference in Privacy-Protected Databases. *Proceedings of the VLDB Endowment*, 11(11):1715–1728, July 2018.

[20] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[21] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2105–2122, New York, NY, USA, 2017. Association for Computing Machinery.

[22] Lukas Burkhalter, Anwar Hithnawi, Alexander Viand, Hossein Shafagh, and Sylvia Ratnasamy. TimeCrypt: Encrypted data stream processing at scale with cryptographic access control. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 835–850, 2020.

[23] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2005.

[24] David Cerdeira, Nuno Santos, P. Fonseca, and S. Pinto. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1416–1432, 2020.

[25] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. *IACR Cryptol. ePrint Arch.*, page 570, 2018.

[26] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Cryptε: Crypto-Assisted Differential Privacy on Untrusted Servers. *CoRR*, abs/1902.07756, 2019.

[27] Intel Corp. Introduction to Intel SGX Sealing. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html.

[28] Henry Corrigan-Gibbs and Dmitry Kogan. Private Information Retrieval with Sublinear Online Time. *IACR Cryptol. ePrint Arch.*, 2019:1075, 2019.

[29] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 44–75, Cham, 2020. Springer International Publishing.

[30] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[31] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Universal Service-Providers for Private Information Retrieval. *J. Cryptology*, 14:37–74, 2001.

[32] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Oblivious Coopetitive Analytics Using Hardware Enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[33] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. 2004.

[34] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. M2R: Enabling Stronger Privacy in MapReduce Computation. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 447–462, Washington, D.C., August 2015. USENIX Association.

[35] Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 523–535, 2017.

[36] Samuel Dooley, Dana Turjeman, John P Dickerson, and Elissa M. Redmiles. Field evidence of the effects of privacy, data transparency, and pro-social appeals on covid-19 app attractiveness. In *CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[37] Morris Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015-08-04 2015.

[38] Saba Eskandarian and Matei Zaharia. ObliDB: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, 2019.

[39] Ben Fisch, Dhinakaran Vinayagamurthy, Dan Boneh, and Sergey Gorbunov. Iron: functional encryption using intel sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 765–782, 2017.

[40] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. HardIDX: Practical and secure index with SGX. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 386–408. Springer, 2017.

[41] Oded Goldreich and Rafail Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.

[42] Michael T. Goodrich. Data-Oblivious External-Memory Algorithms for the Compaction, Selection, and Sorting of Outsourced Data. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA 2011, pages 379–388, New York, NY, USA, 2011. Association for Computing Machinery.

[43] Alexey Gribov, Dhinakaran Vinayagamurthy, and Sergey Gorbunov. StealthDB: a scalable encrypted database with full SQL query support. *CoRR*, abs/1711.02279, 2017.

[44] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. Why Your Encrypted Database Is Not Secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 162–168, New York, NY, USA, 2017. Association for Computing Machinery.

[45] Peeyush Gupta, Yin Li, Sharad Mehrotra, Nisha Panwar, Shantanu Sharma, and Sumaya Almanee. Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. *Proceedings of the VLDB Endowment*, 12(9):1030–1043, May 2019.

[46] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols?

[47] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, number 1, pages 331–335, 2011.

[48] Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening Semi-honest Protocols with Dual Execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284, San Francisco, CA, USA, May 2012. IEEE.

[49] John Kelsey, Shu jen Chang, and Ray Perlner. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, 2016-12-22 2016.

[50] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[51] OASIS labs. A better way to Contact Trace, Part I. https://medium.com/oasislabs/a-better-way-to-contact-trace-7beb12889017.

[52] OASIS labs. A better way to Contact Trace, Part II. https://medium.com/oasislabs/a-better-way-to-contact-trace-part-ii-code-to-back-it-up-50046c4fa6e1.

[53] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to Build a Trusted Database System on Untrusted Storage. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI 2000, USA, 2000. USENIX Association.

[54] Justin Manweiler, Ryan Scudellari, and Landon P. Cox. SMILE: Encounter-Based Trust for Mobile Social Services. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 246–255, New York, NY, USA, 2009. Association for Computing Machinery.

[55] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel Secure Computation Made Easy. In *2015 IEEE Symposium on Security and Privacy*, pages 377–394, 2015.

[56] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A Survey of Published Attacks on Intel SGX, 2020.

[57] Olga Ohrimenko, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma. Observing and Preventing Leakage in MapReduce. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1570–1581. ACM, 2015.

[58] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, Austin, TX, August 2016. USENIX Association.

[59] Femi G. Olumofin and Ian Goldberg. Privacy-Preserving Queries over Relational Databases. In Mikhail J. Atallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*, volume 6205 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2010.

[60] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, pages 587–602, 2016.

[61] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steven M. Bellovin. Blind Seer: A Scalable Private DBMS. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 359–374. IEEE Computer Society, 2014.

[62] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *International conference on the theory and application of cryptology and information security*, pages 250–267. Springer, 2009.

[63] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[64] Christian Priebe, Kapil Vaswani, and Manuel Costa. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 264–278. IEEE Computer Society, 2018.

[65] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. *Mycelium: Large-Scale Distributed Graph Queries with Differential Privacy*, page 327–343. Association for Computing Machinery, New York, NY, USA, 2021.

[66] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: Large-scale Differentially Private Aggregation Without a Trusted Core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP'19)*, October 2019.

[67] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce. Orchard: Differentially Private Analytics at Scale. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, November 2020.

[68] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 38–54, USA, 2015. IEEE Computer Society.

[69] ARM Developer Site. ARM confidential compute architecture (cca). https://developer.arm.com/architectures/architecture-security-features/confidential-computing.

[70] Intel Developer Site. Intel Trust Domain Extensions (Intel TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html.

[71] Lillian Tsai, Roberta De Viti, Matthew Lentz, Stefan Saroiu, Bobby Bhattacharjee, and Peter Druschel. EnClosure: Group Communication via Encounter Closures. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, pages 353–365, New York, NY, USA, 2019. Association for Computing Machinery.

[72] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. SoK: Fully homomorphic encryption compilers. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1092–1108, 2021.

[73] Radek Vingralek. GnatDb: A Small-Footprint, Secure Database System. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*, pages 884–893. Morgan Kaufmann, 2002.

[74] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical Private Queries on Public Data. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 299–313. USENIX Association, 2017.

[75] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS 2015, pages 850–861, New York, NY, USA, 2015. Association for Computing Machinery.

[76] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016. Accessed: 2020-05-27.

[77] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. Cryptology ePrint Archive, Report 2017/030, 2017. https://eprint.iacr.org/2017/030.

[78] Lucie White and Philippe van Basshuysen. Without a trace: Why did Corona apps fail? *Journal of Medical Ethics*, 2021.

[79] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast Extension for coRRElated oT with small communication. Cryptology ePrint Archive, Report 2020/924, 2020. https://eprint.iacr.org/2020/924.

[80] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[81] Samee Zahur, Mike Rosulek, and David Evans. Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits using Half Gates. Cryptology ePrint Archive, Report 2014/756, 2014. https://eprint.iacr.org/2014/756.

[82] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298, Boston, MA, March 2017. USENIX Association.

## Appendix A. Community approval process

As discussed in § 5, CoVault relies on a community approval process to justify trust in the system. Specifically, CoVault relies on community approval for two purposes: (i) to justify the trust in the PSs, which form CoVault's root of trust by attesting and provisioning DPs; (ii) to justify data sources' trust that the query classes to which they contribute their data do what their specification says it does. In both cases, data sources and users can rely on experts they trust who have attested the PSs or reviewed the implementation of PSs and DPs.

Any interested community member can inspect the source code of each system component, verify their measurement hashes, remotely attest the PSs, and publish a signed statement of their opinion. To do so, a witness performs the following operations: 1) Check if another witness it trusts has already verified and attested the PSs (it would not scale to have *every* witness remotely attesting the PSs); if not, inspect the source code of the PSs and the build chain used to compile all system components, make sure they correctly implement the system's specification, and verify that the measurement hash recorded in the system configuration matches the source code, and remotely attest the PSs accordingly; 2) Obtain the current system configuration from the PSs; 3) Review the specification and source code of each query in a given class, and verify the measurement hash of the class's DPs recorded in the configuration; 4) Publish a signed statement of their (the witness's) assessment.

We note that attackers could remove or suppress witness statements, and false witnesses could post false statements about PS attestations or PS and DP (query class) reviews. However, this amounts at most to denial-of-service (DoS) as long as data sources are not distracted by reviews (positive or negative) from witnesses they do not fully trust. Furthermore, we note that the state of the PSs could be rolled back by deleting their sealed states or replacing the sealed states with an earlier version. However, this also amounts to DoS, as it would merely have the effect of making inaccessible some recently defined query classes and their data.

## Appendix B. Ingress processing of data with public attributes

Many analytics applications like epidemic or financial transaction analytics rely on (spatio-)temporal data. In these applications, it is likely that the queries analyse data in a given (space-)time period. If the (spatio-)time attribute reveals no sensitive information, we can exploit data locality: grouping and storing together data according to their public (spatio-)temporal information speeds up the queries that fetch a whole group for processing. To exploit data locality, CoVault creates a materialized view(s) indexed by a public attribute(s): e.g., a public, coarse-grained (space-)time region. In fact, CoVault is a read-only platform, but supports appends to the db in order of a public attribute(s): CoVault's ingress processing pipeline allows incremental data upload from several data sources and produces materialized views securely and efficiently. In this section, we explain CoVault's ingress processing, which is implemented in 2PC by two dedicated DPs called the two Ingress Processors (IPs).

**Batch append** Data sources upload new data in batches, which are padded to obfuscate the exact amount of data being uploaded. Data sources may pre-partition data according to public attributes, or locally pre-join or pre-select data prior to upload. Once a batch is uploaded, the batch becomes immutable and queryable. Before appending that batch to CoVault's db, the IPs can perform pre-processing operations in 2PC; for instance, the IPs can buffer different batches and run group or sort operations (within 2PC) in order to produce or append to materialized views. The actual operations and materialized views are application-specific; in § C, we discuss the example scenario of epidemics analytics. Note

that data sources can contribute data to one or more query classes. CoVault's IP pairs are query-class specific.

**Ingress security and integrity** Data sources upload batches to IPs using session keys, thus are not identifiable by the IPs. The batches are padded, and the padding is "revealed" only within 2PC: no single IP party can determine the actual amount of data in any batch. For added security, data sources can split a single batch into multiple batches randomly and upload them with different session keys. They may also use VPN/Tor to obfuscate their Internet addresses while uploading.

**Storing data in garbled form** As discussed in § 6.3, computing MACs in 2PC is expensive. So, we eliminate the MACs between IPs and DPs as well, by storing values in CoVault's db directly in garbled form. Doing so significantly increases the size of stored data: garbling codes each bit in a 128-bit space. However, this choice eliminates the need for the IPs to compute MACs within 2PC, and for the DPs to verify them. (However, recall that IPs need to *verify* within 2PC the MACs added to their shares by the data sources, and this is unavoidable). This optimization of storing garbled values can be generalized to different query classes by using a separate secret to garble circuits for each query class: an IP pair garbles every datum once for the query class it manages, and each DP pair gets access to the secret of its query class only.

# Appendix C. Further Details and Evaluation of Our Example Epidemics Analytics Scenario

The staggering human, social, and economic cost of the COVID-19 pandemic has led researchers across the sciences to try and build tools that can assist with the containment and mitigation of the virus' spread. Analytics of user mobility, contact and location data could provide insight into how and why a pandemic is spreading, with potentially tremendous benefits to society. But legitimate and serious privacy concerns led most countries to forego the use of such information in the context of COVID-19 [78]. Interestingly, the tension between analytics of personal data and the privacy of data sources is not inherent: Most relevant analytics queries produce statistics (e.g., the basic reproduction number $R_0$ of an epidemic), which do not reveal information about individuals; in many cases, they can be made even differentially private without significant loss of utility.

The fundamental problem, instead, is the lack of a scalable analytics platform that can be trusted to aggregate, store, and use sensitive personal data only for approved analytics. This is precisely the kind of situation for which CoVault is designed. Accordingly, we evaluated CoVault in the context of epidemic analytics (§ 7). In this appendix, we give some supplemental information on CoVault's design to contextualize more in detail our evaluation.

## C.1. Database

Epidemic analytics operates on a time series of individual locations and pairwise contacts among data sources' devices. Such data may originate, for instance, from smartphone apps that record GPS coordinates and pairwise Bluetooth encounters [54], [71], or from a combination of personal devices and Bluetooth beacons installed in strategic locations [9], [14]. We assume that data sources consent to consider as *public* coarse-grained spatio-temporal information related to the data they upload, as well as the inputs and the results of statistical epidemiological queries.

Here, we describe the records and materialized views used in our epidemic analytics scenario. Each view consists of a variable number of records of the form described in Figure 7. There is a separate view for every query class. The views include encounters identified by an encounter id ($eid$). If two data sources share an encounter, they report the same $eid$ for that encounter, along with their own device id $did$. Only mutually confirmed encounters – encounters that were reported by both peers with consistent information on the encounter – are used for analytics. For this purpose, the IPs perform a join of the individual encounter reports, and add a *validity* attribute to potentially mark an encounter as confirmed (see § C.2). Furthermore, the records may report $time$, which is a fine-grained temporal information indicating when an encounter begins, $duration$, which refers to the duration of the encounter, and $aux$, which represents any additional information not currently used in the queries we evaluate in § 7 (e.g., fine-grained locations or signal strength). Finally, note that the records in $T_P$ are organized by device trajectories: these records are indexed by $did1$ and $time$, and each entry contains the indexes of the previous ($prev$) and next ($next$) encounters of $did1$. This view is used by queries that search trajectories in an encounter graph (q3 in Figure 4 and q4 in Figure 9).

**Space-time views** These materialized views speed up queries that have no data-dependent flow control. The views are grouped in space-time regions, indexed by a *public*, coarse-grained space time index (coarse-grained spatio-temporal information is *public* in this setting). Space-time views contain either encounter records ($T_E$) or information related to sick people ($T_S$). When a data source uploads a batch of encounters, it pre-selects the coarse-grained space-time index that batch maps to. The IPs collect batches from different data sources, and generate space-time views grouping data in the same space-time region. Using space-time views accelerates queries whose input parameters specify an arbitrary space-time region in input: in fact, the DPs can locally fetch all records whose coarse-grained index falls within the space-time region in input prior to run the query in 2PC.

The resolution of the space-time regions depends on publicly known population density information and mobility patterns at a given location, day of the week, and time of the day, and is chosen so as to achieve an approximately even number of records per region. Each region is padded to its nominal size to hide the actual number of records contained

Figure 7: Materialized views used for epidemic analytics. The first column is a public index.

| View | Record fields | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $T_E$ | *space-time-region* | eid | did1 | did2 | time | duration | aux | validity |
| $T_S$ | *space-time-region* | did1 | aux | period of contagion | | | | |
| $T_P$ | *time-epoch* | did1, time | did2 | duration | prev | next | aux | validity |

in it. (Note that many shards corresponding to locations at sea, in the wilderness, or night time have a predicted size of zero and therefore do not exist in the views.)

**Shuffled encounter views** As discussed in § 6.3, space-time views can be leveraged only if the queries do not require any secret-dependent data access. Otherwise, we use shuffled encounter views, which support our PIR scheme. These views allow running a sequence of basic queries on $T_P$; however, CoVault cannot reveal the *sequence* of space-time regions analysed, as such sequence might reveal data sources' movements in time. Thus, the only public attribute is a conservatively coarse-grained *time* information, which we call a *time epoch*. The primary key is encrypted and *the records within each epoch are randomly shuffled*. The mapping between a record, its encrypted key, and its position within the view can be reconstructed only inside 2PC. The views are produced by the IPs and the records in an epoch are re-shuffled by the DPs after each use in a query, using the protocol of § 6.1.

**Risk encounter view** This materialized view contains encounters that involve a diagnosed patient and took place during the patient's contagion period. Conceptually, it is the result of a join of $T_E$ and $T_S$, computed incrementally during ingress processing and in cooperation with data sources. The view supports efficient queries that focus on potential and actual infections.

We discuss concrete queries running on $T_E$ and $T_P$ in 7.3.

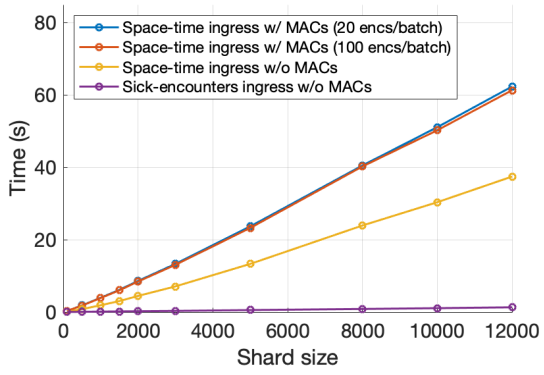
## C.2. Ingress Processing

Next, we discuss CoVault's ingress processing (§ B) specific to epidemic analytics. As mentioned in § C.1, the IPs check whether an encounter is confirmed, and potentially mark it as valid. Here, we give more details about this computation. Before uploading data to a view, the data source partitions its encounters into *per-space-time-region* batches, sorts each batch by $eid$, and *randomly pads* each batch to hide the actual number of encounters in the batch (see § B). Then, the data source secret-shares and MACs the batches following the protocol of § 4.4, and uploads the batches to the two IPs using session keys.

Throughout a day, each IP pair receives batches from data sources and stores them locally (outside 2PC). Periodically, e.g., once per day, each IP pairs run a 2PC, which:

- consumes all per-device batches for the space-time region
- reconstructs the batches from the shares
- verifies the per-batch MACs

Figure 8: Cost of ingress processing on a single space-time region as a function of region size




- merges the (sorted) batches into a space-time region buffer using oblivious sorted merge [46]
- and truncates or pads the sorted list to the expected space-time region size.

The sorting puts padding uploaded by the data sources at the end of the buffer, so truncating the buffer deletes padding first.

Next, the IPs perform a 2PC to confirm encounters. For each $eid$, they check if both peers have uploaded the encounter with consistent locations and times, and set the validity bit of each encounter accordingly. This requires a linear scan of the space-time region. Finally, each IP in the pair appends the space-time region (in garbled form; see § B) to the appropriate tables and materialized views, performing random shuffling when needed.

## C.3. Evaluation of Ingress Processing

Ingress processing (§§ B and C.2) converts batches of records uploaded by data sources to tables/views used by queries. Again, we report the costs of only one of the two circuits of DualEx because ingress does not perform DualEx's equality check and runs the two circuits of completely independently in parallel (the tables/views created by the IPs are stored in their in-circuit, garbled representation in a db, § 6.3).

The two highest lines of Figure 8 – which nearly overlap – show the average time for ingress processing to generate a single space-time region in $T_E$, as a function of the region size (x-axis) and the upload batch size (20 encs/batch or 100 encs/batch) on a pair of cores (i.e., a pair of IPs). The processing performs all the steps mentioned in § C.2. The

costs are slightly super-linear because the IPs sort each batch and merge the sorted batches.

A significant part of this cost – between 40% and 65% depending on the region size – is the verification of per-batch MACs (*cf.* the third line, which shows the cost without MAC verification). This high cost of MAC verification in 2PC is why we store tables/views in garbled form and avoid verifying MACs again in query processing. Note that the cost of ingress processing without MAC verification depends on the region size but not the batch size; in fact, in this case, the batch size impacts only the number of batches and the time to upload each batch to the IPs via a different, secure connection. This cost is negligible compared to the costs of 2PC operations.

The lowest line of Figure 8 is the cost of ingress operations to populate the table of data sources diagnosed sick ($T_S$). This cost is much lower than that of generating a space-time region since $T_S$ is not sorted and each record in $T_S$ has fewer bits, which reduces MAC verification time.

**Scaling** The relevant performance metric for ingress processing is throughput: We want to determine how many core pairs we need to keep up with the rate of data generated by a given administrative entity (e.g., country, city, town). To this end, we run a 5h experiment where $m$ core pairs generate space-time regions containing 100 records each from uploads in batches of size 20. We measured the throughput of ingress processing (in encounters processed per hour), varying $m$ from 1 to 8.

As expected, the results show perfect linear scaling with the number of available core pairs, from $584k$ to $4.75M$ encounters processed per hour for 1 and 8 core pairs, respectively. From this, we can extrapolate the number of core pairs needed to keep up with all encounters generated in a given world region. For example, very conservatively assuming 200 encounters/person/day in urban areas and 100 encounters/person/day in rural areas, we estimate that, every 24h: (a) a mid-sized country with population 80M would generate 11.85B encounters; (b) a big metropolitan area with population 8M would generate 1.6B encounters; (c) a urban city with population 3M would generate 600M encounters; (d) a urban town with population 200k would generate 40M encounters.

Extrapolating from the linear scaling above, we estimate that the two runs of DualEx will need a total of 1660, 226, 86, and 6 core pairs, respectively, for ingress processing. According to these estimates, the ingress processing of a whole country is well within the means of a small-scale data center, while that of a town would require a single pair of machines.

**Power overhead** The Thermal Design Power (TDP), which is the maximum continuous power, consumed by the 8 hardware core Intel Intel(R) Xeon(R) Gold 6244 CPU is 150W, and is 225W for the 32 hardware core AMD EPYC 7543 CPU used in our experiments. A 100,000 core pair deployment of such a system, sufficient for the full ingress processing of 60.25 of such countries with 80M people each (4.82B people in total, more than half of the world's population) on a running basis, would consume 2.57MW, which is 0.011% the total power consumed by data centers globally in 2018 [3]. We believe this cost is justifiable for applications like epidemic analytics where the return to society in terms of human lives saved (through faster and better understanding of an emerging pandemic) is enormous.

Figure 9: Additional query q4. The selection on the public attribute `epoch` is done outside 2PC using the public index of $T_P$.

**(q4)** Count #devices in `B` that encountered a device which *previously* encountered a device in `A`, with both encounters in the time interval `[start,end]`

```
WITH TT AS
 (SELECT * FROM T_P
  WHERE start < time-epoch < end)
SELECT COUNT(DISTINCT(T2.did2))
FROM (TT AS T1) JOIN (TT AS T2)
 ON T1.did2 == T2.did1
WHERE T1.did1 ∈ A AND T2.did2 ∈ B AND
 start < T1.time < T2.time < end
```

## C.4. Evaluation of an additional data-dependent query

This section describes another data dependent query q4, which we also evaluate for latency. The query is shown in Figure 9. This query asks how many devices from set `B` encountered a device from the set `A` indirectly through an intermediate device (all within a given time interval). This query can be used to determine if two outbreaks of an epidemic (corresponding to the sets `A` and `B`) are indirectly connected over 1-hop.

**Latency evaluation** We implemented and evaluated the latency of q4 in the same setting as q3 (§ 7.3). Unlike q3, which makes one traversal over the trajectories of devices in `B`, q4 uses two traversals. One traversal collects all encounters of devices in `A` moving forwards in time; the second traversal collects all encounters of devices in `B` moving backwards in time. We then sort the collected encounters of $\texttt{A} \cup \texttt{B}$ ascending by time, and make a linear pass over them maintaining two Bloom filters, one of all devices that have encountered a device in `A` and the other of devices in `B` who have encountered a device *already* in the first Bloom filter. The result of the query is the size of the second filter at the end.

In our experiments, Bloom filter operations take 709ms per encounter and there is a one-time setup cost of 129ms. Sorting the encounter list of $\texttt{A} \cup \texttt{B}$ has negligible cost in comparison. Assuming $e = 336$ time-epochs, $n = 30$ encounters/person/h, $|\texttt{A}| = 10$ and $|\texttt{B}| = 10$ (these are the same values that we assumed for query q3), this encounter list has length $m = e \cdot n \cdot (|\texttt{A}| + |\texttt{B}|) = 201,600$. The total query latency is $129 + 27 \cdot e \cdot n \cdot (|\texttt{A}| + |\texttt{B}|) + (10 + 709) \cdot m$ = 1.74 days. In this case, the savings over a naive approach of

scanning all encounter records are even more pronounced than those in the case of query q3 (§ 7.3), since the naive approach would have to first sort 165.9B records in 2PC and then scan them maintaining the same two Bloom filters.

## Appendix D. Estimation of end-to-end query latency

We describe how we estimate the end-to-end latency of a query executed in 2PC using our map-reduce approach, as a function of the number of available machine pairs. By reversing the estimation function, we can also easily determine the number of machines needed to attain a given query latency.

We start with two basic mathematical facts that we need for our estimates.

**Lemma 1.** *Given $n$ i.i.d. random variables $X_1, \ldots, X_n$ with normal distributions of mean $\mu$ and standard deviation $\sigma$, let $X = \max(X_1, \ldots, X_n)$. Then, $\mathbb{E}[X] \leq \mu + \sigma\sqrt{2\ln(n)}$.*

*Proof.* This is a folklore result. We provide a simple proof here. Let $t > 0$ be a parameter. Since $e^y$ is a convex function of $y$, by Jensen's inequality, $e^{t\mathbb{E}[X]} \leq \mathbb{E}[e^{tX}] = \mathbb{E}[\max_i e^{tX_i}] \leq \mathbb{E}[\sum_{i=1}^n e^{tX_i}] = \sum_{i=1}^n \mathbb{E}[e^{tX_i}]$. From the moment generating function of the normal distribution, $\mathbb{E}[e^{tX_i}] = t\mu + \frac{1}{2}\sigma^2t^2$. Hence, $e^{t\mathbb{E}[X]} \leq ne^{(t\mu+\frac{1}{2}\sigma^2t^2)}$, and $\mathbb{E}[X] \leq \frac{\ln(n)}{t} + \mu + \frac{t\sigma^2}{2}$. The function on the right is minimized for $t = \frac{\sqrt{2\ln(n)}}{\sigma}$, and its minimum value is $\mu + \sigma\sqrt{2\ln(n)}$, as required. □

In the sequel, we let $\mathbb{E}[D; n]$ denote the expected value of the maximum of $n$ i.i.d. random variables, each drawn from the normal distribution $D$. Lemma 1 says that

$$\mathbb{E}[D; n] \leq \mu + \sigma\sqrt{2\ln(n)}$$

where $\mu$ and $\sigma$ are the mean and standard deviation of $D$.

**Lemma 2.** *For any natural number $K \geq 0$,*

$$\sqrt{K} + \sqrt{K-1} + \ldots + \sqrt{1} \leq \frac{2}{3}((K+1)^{3/2} - 1)$$

*Proof.*

$$\begin{aligned}
& \sqrt{K} + \sqrt{K-1} + \ldots + \sqrt{1} \\
= & \int_K^{K+1} \sqrt{K}\,\mathrm{d}x + \ldots + \int_1^2 \sqrt{1}\,\mathrm{d}x \\
\leq & \int_K^{K+1} \sqrt{x}\,\mathrm{d}x + \ldots + \int_1^2 \sqrt{x}\,\mathrm{d}x \\
= & \int_1^{K+1} \sqrt{x}\,\mathrm{d}x \\
= & \tfrac{2}{3}((K+1)^{3/2} - 1)
\end{aligned}$$

□

Now we explain our estimation of end-to-end latency. Suppose we have $M$ available machines pairs and each machine has $2C$ available cores. This gives us a total of $C$ *units of DualEx execution* per machine pair and a total of $MC$ units of DualEx execution. In the sequel, we use the term *unit* to mean "a unit of DualEx execution".

For a given query, let the queried table have $N$ records. We divide the records of the table into *chunks* of $t$ records each, and then divide the chunks evenly among the $MC$ units. So, each unit starts with $N_c$ chunks, where

$$N_c = \frac{N}{tMC}$$

The best value of $t$ is determined empirically: If $t$ is too small, each mapping and initial reducing circuit does very little work (and setup costs dominate). If $t$ is too high, the state of all parallel cores on a machine may not fit in memory. For tables in our evaluation, we usually pick $t$ to be 10,000 records.

The query actually executes in 4 fine-grained stages. All but the last stage do *not* perform the final equality check of DualEx, but each stage does run the two symmetric DualEx 2PC computations.

1) (Unit-level) Each unit (2+2 cores on a machine pair) maps two chunks out of its $N_c$ chunks and then reduces them to an intermediate result. Each unit then alternates mapping a new chunk and reducing the mapped chunk with the intermediate result previously available on the unit, producing a new intermediate result. This second step is repeated $N_c - 2$ times on each unit till all chunks are consumed and one intermediate result is obtained on each unit. All $MC$ units do these computations in parallel. Let $D_{mmr}$ be the latency distribution of the first two maps followed by reduce on a single unit, and let $D_{mr}$ be the latency distribution of each of one subsequent map+reduce on a single unit.
2) (Machine-pair-level) All $C$ units on a single machine pair reduce their results to a single result using a balanced reduction tree. All $M$ machine pairs do this independently in parallel. Let $D_{mach}$ be the latency distribution of this on a single machine.
3) (Cross-machine) All intermediate results across all machine pairs are reduced using a balanced reduce tree to two final results, which are on a single machine. Only one unit per machine pair is used. At the end, one machine ends up with two results. Let $D_{cmr}$ be the latency distribution of the following cross-machine computation: Two pairs of machines reduce in parallel and one pair sends its result to the other pair at the end. We call this a cross-machine reduction unit (*cmru*).
4) (Final reduce) The machine pair getting the last two intermediate results performs one final reduce with the DualEx equality check at the end. Let the latency distribution of this final step be $D_{fr}$.

We empirically estimate $D_{mmr}$, $D_{mr}$, $D_{mach}$, $D_{cmr}$, and $D_{fr}$ by observing the means and standard deviations of the corresponding latencies while running a query on a small number of machines. Let $\mu_{mmr}$ and $\sigma_{mmr}$ denote the mean and standard deviations of $D_{mmr}$, and similarly for the remaining four distributions. For our end-to-end latency

estimate we model each of these distributions as a *normal* distribution with the measured mean and standard deviation.

There is no synchronization at the end of each stage in our implementation, e.g., if all units on a machine pair have finished stage 1, that machine pair goes ahead with stage 2 without waiting for other machine pairs to finish stage 1. However, we estimate the end-to-end query latency *conservatively* by instead calculating the latency of a hypothetical execution model where there is a full, instantaneous synchronization at the end of each of the first three stages. This latter latency is definitely a conservative upper bound on the actual latency, and is much easier to calculate.

Let $\text{cost}_1$–$\text{cost}_4$ be the expected costs of the four stages above in the conservative (synchronizing) model. We upper-bound each of these separately.

**Estimating $\text{cost}_1$** The latency of stage 1 on each unit has a normal distribution given by $D_1 = D_{mmr} + (N_c - 2)D_{mr}$. From standard properties of normal distributions, $D_1$ has mean and standard deviation $\mu_1$ and $\sigma_1$ where

$$\begin{aligned} \mu_1 &= \mu_{mmr} + (N_c - 2)\mu_{mr} \\ \sigma_1 &= \sqrt{\sigma_{mmr}^2 + (N_c - 2)\sigma_{mr}^2} \end{aligned}$$

Then, since stage 1 consists of $MC$ parallel units, we have:

$$\text{cost}_1 = \mathbb{E}[D_1; MC] \leq \mu_1 + \sigma_1\sqrt{2\ln(MC)}$$

**Estimating $\text{cost}_2$** In stage 2, $M$ machines operate in parallel, each with latency $D_{mach}$. Thus, we have:

$$\text{cost}_2 = \mathbb{E}[D_{mach}; M] \leq \mu_{mach} + \sigma_{mach}\sqrt{2\ln(M)}$$

**Estimating $\text{cost}_3$** Stage 3 is a tree-shaped cross-machine reduce. The number of levels in this tree is $K$ where:

$$K = \lceil \log_2(M) \rceil$$

At the first level, we have $\lceil M/4 \rceil$ parallel cmrus. At the second level, we have $\lceil M/8 \rceil$ parallel cmrus, and so on, till we have only 1 cmru. Hence, we get:

$$\begin{aligned} \text{cost}_3 &\leq \sum_{i=2}^{K} \mathbb{E}[D_{cmr}; \lceil M/(2^i) \rceil] \\ &\leq \sum_{i=2}^{K} \left( \mu_{cmr} + \sigma_{cmr}\sqrt{2\ln\left(\lceil M/(2^i) \rceil\right)} \right) \\ &= (K-1)\mu_{cmr} + \sigma_{cmr}\sqrt{2\ln 2}\sum_{i=2}^{K}\sqrt{\left(\lceil M/(2^i) \rceil\right)} \\ &\leq (K-1)\mu_{cmr} + \sigma_{cmr}\sqrt{2\ln 2}\sum_{i=1}^{K-2}\sqrt{i} \\ &\leq (K-1)\mu_{cmr} + \tfrac{2}{3}\sigma_{cmr}\sqrt{2\ln 2}\left((K-1)^{3/2} - 1\right) \end{aligned}$$

**Estimating $\text{cost}_4$** This cost is immediate:

$$\text{cost}_4 = \mu_{fr}$$

Our computed upper-bound on the query latency is then $\text{cost}_1 + \text{cost}_2 + \text{cost}_3 + \text{cost}_4$.

# Appendix E. Proofs of Security

In this section, we prove the security of CoVault. CoVault combines many building blocks and therefore we slowly build up to the security proof in §E.5.1 and define several intermediate building blocks and properties. Figure 10 gives and outline of how these building blocks fit together.

In §E.1, we introduce notation and the preliminaries necessary for our proofs of security: message authentication codes and their properties (§E.1.1) and secure secret sharing (§E.1.2). In §E.2, we combine MACs and secret sharing to give a construction of a secure secret sharing scheme and a proof of its security. In §E.3, we modify DualEx, a symmetric protocol with one-bit leakage, to enable asymmetric outputs, then prove that this modified protocol (asymmetric DualEx) is secure.

Next, we combine these two building blocks and show that the resulting protocol securely computes a function on a reconstructed input while maintaining output integrity. This follows from our proof in §E.4 that a secure secret sharing is secure for computing on authenticated data (Def. 12) even in the presence of one-bit leakage. Finally, in §E.5, we define CoVault's ideal functionality and provide a proof of end-to-end security for the system.

## E.1. Notation and preliminaries

For any deterministic functionality $\mathcal{H}$, let $\mathcal{H}^-$ be the functionality that computes $\mathcal{H}$ with one-bit leakage. That is, for a malicious party $P_i$, the two-party functionality $\mathcal{H}^-$ returns the same output as $\mathcal{H}$ along with some one-bit leakage function $\ell$ of $P_i$'s choice applied to the other party's input $x$ ($|\ell(x)| = 1$). For a randomized functionality, $\ell$ can depend on the other party's input and the randomness used by the functionality.

Throughout this appendix, we assume that some algorithms can output a distinguished symbol $\perp$. Our security proofs consider PPT (probabilistic polynomial-time) adversaries $\mathcal{A}$. As is standard in cryptography, the security properties of the schemes we introduce are proven by showing that the *advantage* (defined separately for every property) of a PPT adversary $\mathcal{A}$ is upper-bounded by a *negligible function*:

**Definition 1** (negligible function)**.** *A function $f$ is negligible if for all $c \in \mathbb{N}$ there exists an $N \in \mathbb{N}$ such that $f(n) < n^{-c}$ for all $n > N$.*

**E.1.1. Message authentication codes.** In CoVault, we use MACs to provide integrity for the data sent by a device to the system. When a device generates data, it secret-shares it before sending one share to each of the two TEEs. Because standard, additive secret sharing schemes are malleable, we utilize message authentication codes (MACs) to add integrity.

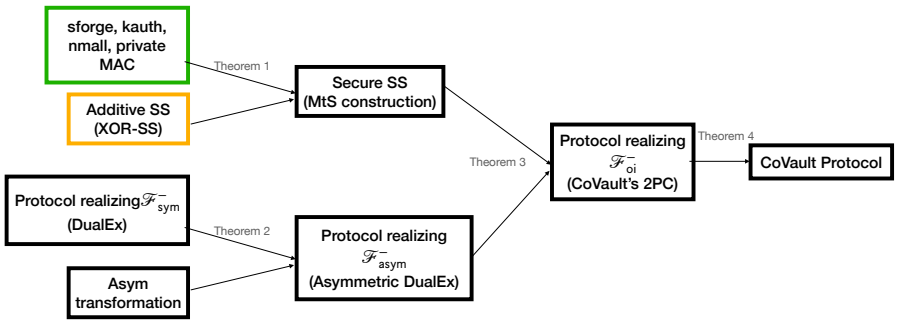



Figure 10: An overview of the proof of security of CoVault.

**Definition 2** (message authentication code (MAC))**.** *A message authentication code (MAC) is a triple of polynomial-time algorithms* $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *such that*

- $\mathsf{KeyGen}$ *takes as input a security parameter* $1^\kappa$ *and outputs a random key* $k \leftarrow_\$ \mathcal{D}_\kappa$
- $\mathsf{Mac}$ *takes as input a key* $k$ *in some domain* $\mathcal{D}_\kappa$ *associated with a security parameter* $1^\kappa$ *and a message* $m$ *in some domain* $\mathcal{D}_m$ *and outputs a tag* $t$*.*
- $\mathsf{Verify}$ *takes as input a key* $k \in \mathcal{D}_\kappa$*, a message* $m \in \mathcal{D}_m$*, and a tag* $t$ *and outputs one of two distinguished symbols* $\top, \bot$*.*

*For correctness, we require that for all* $m \in \mathcal{D}_m$ *and* $k \in \mathcal{D}_\kappa$*,* $\mathsf{Verify}(k, m, \mathsf{Mac}(k, m)) = \top$*.*

*Notation. Define* $\mathsf{Mac}_k(m) := \mathsf{Mac}(k, m)$ and $\mathsf{Verify}_k(m, t) := \mathsf{Verify}(k, m, t)$.

The security guarantees of MACs span a wide range. We define the relevant notions below.

*One-time strong unforgeability. Informally, one-time strong unforgeability says that it is infeasible to forge a tag on a message without knowing the key (including a new tag on a message for which the attacker already knows a tag).

Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to $M$ as $\mathsf{MAC1\text{-}sforge\text{-}adv}[\mathcal{A}, M] :=$

$$\Pr\left[\begin{array}{c} m \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ t \leftarrow \mathsf{Mac}_k(m); \\ (m', t') \leftarrow \mathcal{A}(t) \end{array} : \begin{array}{c} (m', t') \neq (m, t) \bigwedge \\ \mathsf{Verify}_k(m', t') = \top \end{array}\right].$$

**Definition 3** (one-time strong unforgeability)**.** *We say a MAC* $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *is* one-time strongly unforgeable *(alternatively,* one-time strongly secure*) if, for all* PPT *adversaries* $\mathcal{A}$*, there exists a negligible function* $negl$ *such that* $\mathsf{MAC1\text{-}sforge\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$*.*

*One-time key authenticity. Standard notions of MAC security deal only with the consequences of an attacker viewing a message-tag pair. In our setting, we introduce a new notion of security for MACs which considers the case in which $\mathcal{A}$ has access to a message-key pair. Informally, a MAC is key authentic if it is difficult to find a new key which still authenticates a given message-tag pair.

Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to $M$ as $\mathsf{MAC1\text{-}kauth\text{-}adv}[\mathcal{A}, M] :=$

$$\Pr\left[\begin{array}{c} m \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ t \leftarrow \mathsf{Mac}_k(m); \\ k' \leftarrow \mathcal{A}(k) \end{array} : \begin{array}{c} k' \neq k \bigwedge \\ \mathsf{Verify}_{k'}(m, t) = \top \end{array}\right].$$

**Definition 4** (one-time key authenticity)**.** *We say a MAC* $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *is* one-time key authentic *if, for all* PPT *adversaries* $\mathcal{A}$*, there exists a nonnegligible function* $negl$ *such that* $\mathsf{MAC1\text{-}kauth\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$*.*

*One-time non-malleability. In this case, we require that an adversary cannot cause a fixed, known tag to verify a message even if it can modify the message and key by some additive shift.

Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to $M$ as $\mathsf{MAC1\text{-}nmall\text{-}adv}[\mathcal{A}, M] :=$

$$\Pr\left[\begin{array}{c} m \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ t \leftarrow \mathsf{Mac}_k(m); \\ (\Delta_m, \Delta_k) \leftarrow \mathcal{A}(t) \end{array} : \begin{array}{c} (\Delta_m, \Delta_k) \neq (0, 0) \bigwedge \\ \mathsf{Verify}_{k+\Delta_k}(m + \Delta_m, t) = \top \end{array}\right].$$

**Definition 5** (one-time non-malleability)**.** *We say a MAC* $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *is* one-time non-malleable *if, for all* PPT *adversaries* $\mathcal{A}$*, there exists a nonnegligible function* $negl$ *such that* $\mathsf{MAC1\text{-}nmall\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$*.*

*Privacy. Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to $M$ as $\mathsf{MAC\text{-}priv\text{-}adv}[\mathcal{A}, M] :=$

$$\left|\Pr\left[\begin{array}{c} (m_0, m_1) \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ b \leftarrow_\$ \{0, 1\}; \\ t_b \leftarrow \mathsf{Mac}_k(m_b) \end{array} : \mathcal{A}(t_b) = b\right] - \frac{1}{2}\right|.$$

**Definition 6** (privacy)**.** *We say a MAC* $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ *is* private *if, for all* PPT *adversaries*

*$\mathcal{A}$, there is a negligible function $negl$ such that* $\mathsf{MAC\text{-}priv\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$.

**E.1.2. Secure secret-sharing.** As mentioned previously, the data sent to CoVault is secret-shared by the source between two TEEs. Basic secret-sharing schemes only deal with privacy of the shared data and do not consider accuracy of reconstruction. In this section, we define (in addition to the notion of privacy) a notion of authenticity which captures the property that the holder of a share of the data cannot, without being detected, change its share in a way that causes the reconstructed data to be altered. A secret sharing scheme with this additional property is called "secure" and will ultimately be achieved by leveraging MACs (§E.2).

**Definition 7** (secret-sharing scheme)**.** *A pair of polynomial-time algorithms $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is a (two-party) secret-sharing scheme if*

- $\mathsf{Share}$ *takes as input a security parameter $1^\kappa$ and a value $x$ in the domain $\mathcal{D}_\kappa$ associated with $\kappa$ (e.g., $\mathcal{D}_\kappa = \{0,1\}^\kappa$) and outputs two shares $\mathsf{sh}_1, \mathsf{sh}_2$. We assume $\kappa$ is implicit in each share.*
- $\mathsf{Rec}$ *takes as input two shares and outputs either a value $y \in \mathcal{D}_\kappa$ or $\perp$.*

*For correctness, we require that for all $\kappa$ and $x \in \mathcal{D}_\kappa$,* $\mathsf{Rec}(\mathsf{Share}(1^\kappa, x)) = x$.

*Privacy. Informally, privacy says that, given a share of one of two values $x_0, x_1$, an adversary $\mathcal{A}$ cannot distinguish whether $x_0$ or $x_1$ was shared.

Let $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ be a secret-sharing scheme. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s *advantage* with respect to $\Sigma$ as $\mathsf{SS\text{-}priv\text{-}adv}[\mathcal{A}, \Sigma] :=$

$$\left| \Pr \left[ \begin{array}{c} (x_0, x_1, i) \leftarrow \mathcal{A}(1^\kappa); \\ b \leftarrow_\$ \{0,1\}; \\ (\mathsf{sh}_{b,1}, \mathsf{sh}_{b,2}) \leftarrow \mathsf{Share}(1^\kappa, x_b) \end{array} : \mathcal{A}(\mathsf{sh}_{b,i}) = b \right] - \frac{1}{2} \right|.$$

**Definition 8** (privacy)**.** *We say a secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is* private *if, for all PPT adversaries $\mathcal{A}$, there is a negligible function $negl$ such that $\mathsf{SS\text{-}priv\text{-}adv}[\mathcal{A}, \Sigma] \leq negl(\kappa)$.*

*Authenticity. Informally, authenticity guarantees that any modification to a share will result in Rec returning $\perp$ with high probability.

Let $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ be a secret-sharing scheme. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to $\Sigma$ as $\mathsf{SS\text{-}auth\text{-}adv}[\mathcal{A}, \Sigma] :=$

$$\Pr \left[ \begin{array}{c} (x, i) \leftarrow \mathcal{A}(1^\kappa); \\ (\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}(1^\kappa, x); \\ \mathsf{sh}_i' \leftarrow \mathcal{A}(\mathsf{sh}_i) \end{array} : \begin{array}{c} \mathsf{sh}_i' \neq \mathsf{sh}_i \ \bigwedge \\ \mathsf{Rec}(\mathsf{sh}_i', \mathsf{sh}_{3-i}) \neq \perp \end{array} \right].$$

**Definition 9** (authenticity)**.** *We say a secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is* authenticated *if, for all PPT adversaries $\mathcal{A}$, there is a negligible function $negl$ such that $\mathsf{SS\text{-}auth\text{-}adv}[\mathcal{A}, \Sigma] \leq negl(\kappa)$.*

**Definition 10** (security)**.** *We say a secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is* secure *if it is both private and authenticated.*

## E.2. Secure secret sharing construction

There are several ways to construct a secure secret-sharing scheme $\Sigma_M = (\mathsf{Share}_M, \mathsf{Rec}_M)$ using a MAC $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ and base (non-authenticated) secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$. We introduce MAC-then-Share, the construction used in CoVault, and analyze its security.

**Definition 11** (MAC-then-share (MtS))**.** *Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC and $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ a secret-sharing scheme. Define $\Sigma_M = (\mathsf{Share}_M, \mathsf{Rec}_M)$, the MtS secret-sharing scheme based on $M$ and $\Sigma$, as follows:*

- $\mathsf{Share}_M$ *takes as input a security parameter $1^\kappa$ and a value $x$ in the domain $\mathcal{D}_\kappa$ associated with $\kappa$. It generates a single key $k$ using* $\mathsf{KeyGen}$ *and uses it to computes one tag $t$ on the secret value $x$ as $t := \mathsf{Mac}_k(x)$. Now it shares $x, k$ by computing $(x_1, x_2) \leftarrow \mathsf{Share}(x)$ and $(k_1, k_2) \leftarrow \mathsf{Share}(k)$, and outputs two shares $\mathsf{sh}_1$, $\mathsf{sh}_2$ with $\mathsf{sh}_i := (x_i, k_i, t)$.*
- $\mathsf{Rec}_M$ *takes as input two shares. If the tags match, it reconstructs $y := \mathsf{Rec}(x_1, x_2)$ and $k' := \mathsf{Rec}(k_1, k_2)$. Then, if $\mathsf{Verify}_{k'}(y, t) = \top$, it returns $y$; otherwise it returns $\perp$.*

What properties are required of the MAC and secret-sharing scheme in order for their composition to be a secure secret-sharing scheme? In Theorem 1, we show that the MAC must meet all four properties presented in Section E.1.1 in order for the MtS construction to be secure.

**Theorem 1.** *If $M$ is a one-time strongly unforgeable, key authentic, non-malleable, and private MAC and $\Sigma$ an additive secret-sharing scheme, then the MtS secret-sharing scheme $\Sigma_M$ constructed from $M$ and $\Sigma$ is secure.*

*Proof.* The privacy of $\Sigma_M$ follows directly from privacy of $M$ (since $\mathsf{Mac}_k(x)$ reveals nothing about $x$) and $\Sigma$ ($x_i$ also reveals nothing about $x$).

Authenticity of $\Sigma_M$ is a bit more unwieldy. We prove the contrapositive that if $\Sigma_M$ is not authenticated, then either (1) $M$ is not strongly secure, (2) $M$ lacks key authenticity, or (3) $M$ is malleable. As before, we do this by reduction of an adversary $\mathcal{A}$ with $\mathsf{SS\text{-}auth\text{-}adv}[\mathcal{A}, \Sigma_M]$ nonnegligible to three adversaries for each of the three corresponding games, and show that at least one of them has nonnegligible advantage in its game.

First, we construct an adversary $\mathcal{A}_{\mathsf{sforge}}$ for the game $\mathsf{MAC1\text{-}sforge}_{\mathcal{A}_{\mathsf{sforge}}, M}$. $\mathcal{A}_{\mathsf{sforge}}$ receives $x, i$ from $\mathcal{A}$ and constructs a MtS triple $\mathsf{sh}_i := (x_i, k_i, t_i)$ as follows: it sends $x$ to its game to get $t := \mathsf{Mac}_k(x)$, picks a random $k_i \in \mathcal{D}_\kappa$, and runs Share on $x, t$ to get $x_1, x_2, t_1, t_2$. Now $\mathcal{A}_{\mathsf{sforge}}$ runs $\mathcal{A}$ on $\mathsf{sh}_i$ to get $\mathsf{sh}_i' := (x_i', k_i', t_i')$ and returns $(x', t')$, where $x' \leftarrow \mathsf{Rec}(x_i', x_{3-i})$ and $t' \leftarrow \mathsf{Rec}(t_i', t_{3-i})$.

Next, we construct an adversary $\mathcal{A}_{\mathsf{kauth}}$ for the game $\mathsf{MAC\text{-}kauth}_{\mathcal{A}_{\mathsf{kauth}}, M}$. $\mathcal{A}_{\mathsf{kauth}}$ is given $x, i$ by $\mathcal{A}$ and sends $x$ to its game, which sends back a key $k$. It uses this key to construct a MtS triple $\mathsf{sh}_i := (x_i, k_i, t)$, computing $t := \mathsf{Mac}_{k_i}(x)$ and running Share on $x, k$ to get $x_1, x_2, k_1, k_2$.

Now $\mathcal{A}_{\mathsf{kauth}}$ runs $\mathcal{A}$ on $\mathsf{sh}_i$ to get $\mathsf{sh}'_i := (x'_i, k'_i, t')$ and returns $k' \leftarrow \mathsf{Rec}(k'_i, k_{3-i})$.

Third, we construct an adversary $\mathcal{A}_{\mathsf{nmall}_M}$ for the MAC non-malleability game.$\mathcal{A}_{\mathsf{nmall}_M}$ is given $x, i$ by $\mathcal{A}$ and sends $x$ to its game to receive $t$. It constructs anMtS triple $\mathsf{sh}_i := (x_i, k_i, t_i)$ by choosing a random $k \in \mathcal{D}_\kappa$ and running $\mathsf{Share}$ on $x, k, t$. Now $\mathcal{A}_{\mathsf{nmall}_M}$ runs $\mathcal{A}$ on $\mathsf{sh}_i$ to get $\mathsf{sh}'_i := (x'_i, k'_i, t')$. It computes $x' \leftarrow \mathsf{Rec}(x'_i, x_{3-i}), k' \leftarrow \mathsf{Rec}(k'_i, k_{3-i})$, and returns $(x', k')$.

We now analyze the winning probability of each of the three adversaries. By our assumption, $\mathcal{A}$ wins its game with nonnegligible probability, implying $t' = t$. If $\mathcal{A}$ returns $\mathsf{sh}'_i$ such that $k'_i = k_i$ with nonnegligible probability, then $\mathcal{A}_{\mathsf{sforge}}$ has nonnegligible advantage and we are done. If not, then with nonnegligible probability, $\mathcal{A}$ returns $\mathsf{sh}'_i$ with $k'_i \neq k_i$. If $x'_i = x_i$ a nonnegligible fraction of the time, $\mathcal{A}_{\mathsf{kauth}}$ has nonnegligible advantage in its game, and we are done. Otherwise, $x'_i \neq x_i$ and $\mathcal{A}_{\mathsf{nmall}_M}$ has nonnegligible advantage in the non-malleability game for $M$: it has a pair $(\Delta_m := x'_i - x_i, \Delta_k := k'_i - k_i)$ such that $\mathsf{Verify}_{k'_i + k_{3-i}}(x'_i + x_{3-i}, t) = \mathsf{Verify}_{(k_i + k_{3-i}) + \Delta_k}((x_i + x_{3-i}) + \Delta_m, t) = \top$.

Thus, if $M$ is authenticated, it is not strongly secure, lacks key authenticity, or is malleable, all of which contradict our assumptions. □

Note that the proof also holds for the XOR secret sharing scheme by substituting $+, -$ for $\oplus$.

**Concrete Construction.** CoVault uses KMAC256[49], a NIST-standardized SHA3-based MAC. It can be abstracted as follows, where $||$ indicates concatenation. (We omit some additional parameters which are constant and public in CoVault; see Appendix A in [49] and Section 6.1 in [37].)

$\mathsf{KeyGen}$: Use $\mathsf{SHA3}$'s key generation algorithm.
$\mathsf{Mac}_k(m)$: Compute $h := \mathsf{SHA3}(k)$ and announce it publicly. Output $t := \mathsf{SHA3}(k||m)$.
$\mathsf{Verify}_k(m, t)$: Output $\top$ iff $\mathsf{SHA3}(k) = h \wedge \mathsf{SHA3}(k||m) = t$.

This MAC meets the conditions of Theorem 1 in the random oracle model (ROM):

- **one-time strong unforgeability:** Due to randomness of the output $\mathsf{SHA3}(k||m')$.
- **one-time key authenticity:** Due to collision-resistance, which guarantees that for $k \neq k'$ we have with overwhelming probability that $\mathsf{SHA3}(k||m) \neq \mathsf{SHA3}(k'||m)$.
- **one-time non-malleability:** Again due to collision-resistance, since for $(k, m) \neq (k', m')$ we have with overwhelming probability that $\mathsf{SHA3}(k||m) \neq \mathsf{SHA3}(k'||m')$.
- **privacy:** Due to randomness of the output of SHA3.

Hence, CoVault's MtS construction of $\Sigma_M$ with $\Sigma$ as the additive secret sharing scheme and $M$ as KMAC256 is secure. In §E.4, we will show that this means it can safely be used inside a secure computation.

## E.3. DualEx functionality with asymmetric outputs

Before turning to the use of our secure secret sharing construction within a secure multi-party (two-party) computation protocol, we must discuss the exact protocol used by CoVault and its non-standard security guarantees.

CoVault uses the DualEx protocol[48] as a building block. DualEx guarantees security against malicious adversaries at almost the cost of of semi-honest protocols, but with the following caveats: it can compute any *symmetric* two-party functionality $\mathcal{F}_{\text{sym}}$, and it does so with one-bit leakage. By symmetric we mean that both parties are required to receive the same output.

In this section, we will address how to modify DualEx to allow for the computation of *asymmetric* functionalities while maintaining the same security guarantees (malicious with one-bit leakage). This modified DualEx, which we dub "asymmetric DualEx", will be used to implement the core functionality of CoVault, and the one-bit leakage will carry through the remainder of the proofs.

Let $f : \mathcal{X} \times \mathcal{X} \to \mathcal{Y}$ be some function. In the CoVault setting, we want the output of $f$ to be shared between the parties so that neither learns the output. More specifically, we want to compute the functionality $\mathcal{F}_{\text{asym}}$ that takes as input $x_1$ from one party and $x_2$ from the other, and returns a uniformly random $r$ to the first party and $f(x_1, x_2) \oplus r$ to the second party. When $P_1$ is honest, $r$ is chosen by $\mathcal{F}_{\text{asym}}$; when $P_1$ is malicious, $P_1$ chooses $r$.

To do this, we construct a two-party protocol $\Pi$ that computes $\mathcal{F}_{\text{asym}}$ via access to the symmetric functionality $\mathcal{F}_{\text{sym}}$ that takes inputs $(x_1, r_1), (x_2, r_2) \in \mathcal{X} \times \mathcal{Y}$ from the parties, computes $y := f(x_1, x_2) \oplus r_1 \oplus r_2$, and returns $y$ to both parties. The protocol proceeds as follows:

- Each party $P_i$ holds an input $x_i \in \mathcal{X}$. It additionally samples a uniform blinding value $r_i \in \mathcal{Y}$. The parties then provide their inputs $(x_1, r_1)$ and $(x_2, r_2)$, respectively, to $\mathcal{F}_{\text{sym}}$.
- Both parties receive in return $y := f(x_1, x_2) \oplus r_1 \oplus r_2$.
- $P_1$ computes its output as $y_1 := r_1$, while $P_2$ computes its output as $y_2 := y \oplus r_2$.

Notice that $y \oplus r_2 = f(x_1, x_2) \oplus r_1$, so the protocol outputs $f(x_1, x_2) \oplus r_1$ to the second party and the parties now hold shares of $f(x_1, x_2)$. Below we show that this modified protocol maintains the same security guarantees as the base DualEx.

**Theorem 2.** $\Pi$ *securely computes* $\mathcal{F}^-_{asym}$ *against malicious adversaries in the* $\mathcal{F}^-_{sym}$*-hybrid model.*

*Proof.* Let $\mathcal{A}$ be a PPT adversary corrupting party $P_i$. To prove the security of $\Pi$ against malicious adversaries in the $\mathcal{F}^-_{\text{sym}}$-hybrid world, we give an adversary $\mathcal{S}_i$ in the ideal world that simulates an execution of $\Pi$ in the hybrid world. We first consider the case of a corrupted $P_1$.

**Simulator $\mathcal{S}_1$:** $\mathcal{S}_1$ has access to the ideal functionality $\mathcal{F}^-_{\text{asym}}$ computing $f(x_1, x_2) \oplus r$ with one-bit leakage. Given f, $\mathcal{S}_1$ works as follows:

- Receive inputs $x'_1, r'_1$, and a leakage function $\ell : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ from $P_1$.
- Sample $y^* \leftarrow_\$ \mathcal{Y}$. Convert $\ell$ into a function $\ell^* : \mathcal{X} \to \{0,1\}$ by letting $\ell^*(x) = \ell(x, y^* \oplus r'_1 \oplus f(x'_1, x))$ for all $x \in \mathcal{X}$.
- Send $(x'_1, r'_1)$ and $\ell^*$ to the ideal functionality $\mathcal{F}^-_{\text{asym}}$; the honest party sends $x_2$ to $\mathcal{F}^-_{\text{asym}}$.
- Receive in return from $\mathcal{F}^-_{\text{asym}}$ $r'_1$ and some one-bit leakage $b^* := \ell^*(x_2)$. The honest party receives $y^*_2 := f(x'_1, x_2) \oplus r'_1$.
- Send $y^*, b^*$ to $P_1$.

We prove indistinguishability of the following distribution ensembles.

*Ideal experiment. This is defined by the interaction of $\mathcal{S}_1$ with the ideal functionality $\mathcal{F}^-_{\text{asym}}$. $\mathcal{A}$ outputs an arbitrary function of its view; the honest party outputs what it received from the experiment, namely $y^*_2$. Let $\text{IDEAL}_{\mathcal{S}_1}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the output of the adversary $\mathcal{S}_1$ and the output of the honest party. Concretely,

$$\text{IDEAL}_{\mathcal{S}_1}(1^\kappa, x_1, x_2, \ell, f) = ((y^*, b^*), y^*_2).$$

*Hybrid experiment. let $\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the view of the adversary $\mathcal{A}$ and the output of the honest party in the $\mathcal{F}^-_{\text{sym}}$-hybrid world. Concretely,

$$\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f) = ((y, b), y_2).$$

$\mathcal{S}_1$ perfectly simulates the view of a malicious $P_1$ in the hybrid world. Because $r_2$ is chosen uniformly at random, both $y^*$ and $y$ are distributed uniformly at random and are thus perfectly indistinguishable. By the definition of $\ell^*$, $b^* = \ell(x_2, y^* \oplus f(x'_1, x_2) \oplus r'_1$. Furthermore, by the definition of $y$, $r_2 = y \oplus f(x'_1, x_2) \oplus r'_1$, so $b = \ell(x_2, r_2)$ is perfectly indistinguishable from $b^*$. Hence the joint distributions of $(y^*, b^*)$ and $(y, b)$ are perfectly indistinguishable. Finally, $y^*_2$ and $y_2$ are identical by the definition of $y_2$ and thus perfectly indistinguishable.

Next, we give a simulator $\mathcal{S}_2$ for the case of a corrupted $P_2$.

**Simulator $\mathcal{S}_2$:** $\mathcal{S}_2$ has access to the ideal functionality $\mathcal{F}^-_{\text{asym}}$ computing $f(x_1, x_2) \oplus r$ with one-bit leakage. Given $f$, $\mathcal{S}_2$ works as follows:

- Receive inputs $x'_2, r'_2$, and a leakage function $\ell$ from $P_2$.
- Forward $x'_2$ and $\ell$ to the ideal functionality $\mathcal{F}^-_{\text{asym}}$; the honest party sends $x_1$ to $\mathcal{F}^-_{\text{asym}}$.
- Receive in return from $\mathcal{F}^-_{\text{asym}}$ the output $z := f(x_1, x'_2) \oplus r_1$ for uniformly random $r_1$ and some one-bit leakage $b := \ell(x_1)$. The honest party receives $r_1$.
- Compute $y^* := z \oplus r'_2$. Send $y^*, b$ to $P_2$.

We prove indistinguishability of the following distribution ensembles.

*Ideal experiment. This is defined by the interaction of $\mathcal{S}_2$ with the ideal functionality $\mathcal{F}^-_{\text{asym}}$. $\mathcal{A}$ outputs an arbitrary function of its view; the honest party outputs its honestly computed value of $y_1$, namely $r_1$. Let $\text{IDEAL}_{\mathcal{S}_2}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the output of the honest party and the output of the adversary $\mathcal{S}_2$. Concretely,

$$\text{IDEAL}_{\mathcal{S}_2}(\kappa, x_1, x_2, \ell, f) = (r_1, (y^*, b)).$$

*Hybrid experiment. let $\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the the output of the honest party and the view of the adversary $\mathcal{A}$ in the $\mathcal{F}^-_{\text{sym}}$-hybrid world. Concretely,

$$\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f) = (r_1, (y, b)).$$

$\mathcal{S}_2$ perfectly simulates the view of a malicious $P_2$ in the hybrid world, since $y^* = z \oplus r'_2 = f(x_1, x'_2) \oplus r_1 \oplus r'_2 = y$.

Therefore, $\Pi$ is secure (up to 1 bit of leakage) against malicious adversaries in the $\mathcal{F}^-_{\text{sym}}$-hybrid model. □

Notice that the DualEx protocol is an instantiation of $\mathcal{F}^-_{\text{sym}}$, so our modified protocol is a real-world protocol with the same security guarantees against malicious parties as the original DualEx: privacy up to one bit of leakage and full correctness of the output.

## E.4. Secure computation on shared data

We are now ready to pull together the two branches of CoVault's approach: secure secret sharing and secure computation with one-bit leakage.

In CoVault, we wish to securely run some function on the data collected from various data sources; as is common, we will achieve this via secure multi-party (two-party) computation. That data is stored in secret-shared form, and must therefore be reconstructed within the secure computation. In this section, we define the desired functionality in this case: one that outputs only either the correct computation $f(x)$ or aborts ($\perp$) (we call this ideal functionality $\mathcal{F}_{\mathsf{oi}}$, for "output integrity"). A secret sharing scheme which, when used to reconstruct within the secure computation, enables this ideal functionality is called *secure for computing on authenticated data*.

Fix some function $f$ and secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$. Define the function $f_\Sigma$ as

$$f_\Sigma(\mathsf{sh}_1, \mathsf{sh}_2) = \begin{cases} \perp & \mathsf{Rec}(\mathsf{sh}_1, \mathsf{sh}_2) = \perp \\ f(\mathsf{Rec}(\mathsf{sh}_1, \mathsf{sh}_2)) & \text{otherwise.} \end{cases}$$

We consider a hybrid-world execution in which a dealer $D$ shares some value $x$ between two parties $P_1$ and $P_2$. At some later point in time, the parties invoke an ideal functionality $\mathcal{F}$ that computes $f_\Sigma$ (with abort). In more detail, the hybrid-world execution with some function $f$ using an input $x$, both potentially chosen by the malicious party, proceeds as follows:

**Initial sharing:** $D$ runs $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}(1^\kappa, x)$ and sends $\mathsf{sh}_i$ to $P_i$.

**Parties invoke ideal functionality:** Party $P_i$ sends an input $\mathsf{sh}_i'$ to the ideal functionality $\mathcal{F}$ computing $f_\Sigma$; an honest party sends the share it received from the dealer. $\mathcal{F}$ computes $y := f_\Sigma(\mathsf{sh}_1', \mathsf{sh}_2')$ and sends $y$ to the malicious party.

**Output delivery:** After receiving $y$, the malicious party tells the ideal functionality to either continue or abort. In the former case, $\mathcal{F}$ sends $y$ to the honest party; in the latter case, it sends $\perp$ to the honest party. The honest party outputs whatever it receives from the ideal functionality.

For an adversary $\mathcal{A}$, let $\text{H}_\mathcal{A}^\Sigma(1^\kappa, x, f)$ be the joint random variable containing the view of the malicious party and the output of the honest party.

We compare a hybrid-world execution to an *ideal-world* execution in which the parties have access to an ideal functionality $\mathcal{F}_{\mathsf{oi}}$ that can (only) return $y = f(x)$ ($\mathsf{oi}$ stands for "output integrity"). Concretely, the ideal-world execution proceeds as follows:

**Parties invoke ideal functionality:** The parties invoke $\mathcal{F}_{\mathsf{oi}}$, which sends $f(x)$ to the malicious party.

**Output delivery:** After receiving $f(x)$, the malicious party tells $\mathcal{F}_{\mathsf{oi}}$ to either continue or abort. In the former case, the ideal functionality sends $f(x)$ to the honest party; in the latter case, it sends $\perp$ to the honest party. The honest party outputs whatever it receives from the ideal functionality; the malicious party can output an arbitrary function of its view.

For an adversary $\mathcal{S}$, let $\text{IDEAL}_\mathcal{S}(1^\kappa, x, f)$ be the joint random variable containing the output of the adversary and the output of the honest party.

**Definition 12.** *We say $\Sigma$ is* secure for computing on authenticated data *if for all* PPT $\mathcal{A}$ *there exists a* PPT $\mathcal{S}$ *such that distribution ensembles $\{\text{H}_\mathcal{A}^\Sigma(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\text{IDEAL}_\mathcal{S}(1^\kappa, x, f)\}_{\kappa,x,f}$ are computationally indistinguishable.*

However, things are not as simple as that: CoVault uses asymmetric DualEx, so we have one bit of leakage to contend with. Specifically, in our case both functionalities $\mathcal{F}$ and $\mathcal{F}_{\mathsf{oi}}$ will leak one bit of information. Next, we will show that this is not a problem and that any secure secret sharing scheme is secure for computing on authenticated data (even when the secure computation leaks one bit of information).

Let $\mathcal{F}^-$ be the ideal two-party functionality that takes as input two secret shares $\mathsf{sh}_1, \mathsf{sh}_2$, computes $y := f_\Sigma(\mathsf{sh}_1, \mathsf{sh}_2)$ (where $f_\Sigma$ is defined as in Section E.4), and outputs $y$ to both parties with one-bit leakage.

Define the following protocol $\Pi$ for computing $f_\Sigma$ with one-bit leakage in the $\mathcal{F}^-$-hybrid model:

1) The dealer $D$ shares some value $x$ as $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}(1^\kappa, x)$ and sends $\mathsf{sh}_i$ to $P_i$.
2) $P_i$ sends an input $\mathsf{sh}_i'$ to $\mathcal{F}^-$; the honest party sends the share it received from the dealer. The malicious party additionally sends a leakage function $\ell$.
3) $\mathcal{F}^-$ computes $y := f_\Sigma(\mathsf{sh}_1', \mathsf{sh}_2')$ and sends $y$ and $\ell$ evaluated on the honest party's share to the malicious party.
4) The malicious party tells $\mathcal{F}^-$ to either continue or abort. In the former case, $\mathcal{F}^-$ sends $y$ to the honest party; in the latter case, it sends $\perp$. The honest party then outputs whatever it received from the ideal functionality.

**Theorem 3.** *If $\Sigma$ is a secure secret-sharing scheme, then it is secure for computing on authenticated data (even in the presence of one-bit leakage).*

*Proof.* Let $\mathcal{A}$ be a PPT adversary corrupting party $P_i$ and let $\mathcal{A}$ fix $x, f$. We prove security by simulation: to prove the security of $\Pi$ against malicious adversaries in the $\mathcal{F}^-$-hybrid world, we give an adversary $\mathcal{S}$ (the simulator) in the ideal world that interacts with $P_i$ and the ideal functionality computing $f$ with one-bit leakage. $\mathcal{S}$ simulates an execution of $\Pi$ in the hybrid world.

**Simulator** $\mathcal{S}$ : Given $1^\kappa$, $\mathcal{S}$ works as follows:

1) $\mathcal{S}$ simulates the dealer by choosing a random value $\hat{x}$ and running $(\hat{\mathsf{sh}}_1, \hat{\mathsf{sh}}_2) \leftarrow \mathsf{Share}(1^\kappa, \hat{x})$. It gives $\hat{\mathsf{sh}}_i$ to $P_i$.
2) $\mathcal{S}$ receives $\hat{\mathsf{sh}}_i'$ and a leakage function $\ell$ from $P_i$.
3) If $\hat{\mathsf{sh}}_i' \neq \hat{\mathsf{sh}}_i$, then $\mathcal{S}$ sends $\mathsf{abort}$ to the ideal functionality computing $f$ with one-bit leakage, gives $\perp$ to $P_i$, and halts. Otherwise, $\mathcal{S}$ sends a new leakage function $\ell^* : \mathcal{X} \to \{0,1\}$, where $\ell^*(x) := \ell(x \oplus \hat{\mathsf{sh}}_1)\ \forall x \in \mathcal{X}$, and receives $y := f(x)$ and a bit $b^* := \ell^*(x)$. $\mathcal{S}$ forwards both values to $P_i$.
4) If $P_i$ aborts, then $\mathcal{S}$ also aborts; otherwise, it sends $\mathsf{continue}$ to the ideal functionality.

We prove indistinguishability of the ideal and $\mathcal{F}^-$-hybrid distribution ensembles using a sequence of experiments.

**Ideal experiment.** This is defined by the interaction of $\mathcal{S}$ with the ideal functionality computing $f$ with one-bit leakage. $\mathcal{A}$ outputs an arbitrary function of its view; the honest party outputs what it received from the experiment. Let $\text{IDEAL}_\mathcal{S}(1^\kappa, x, \ell, f)$ be the joint random variable containing the output of the adversary $\mathcal{S}$ and the output $y_2$ of the honest party, which is either $y$ or $\perp$. Concretely,

$$\text{IDEAL}_\mathcal{S}(1^\kappa, x, \ell, f) = ((\hat{\mathsf{sh}}_1, \hat{\mathsf{sh}}_1'\ell, y, b^*, \mathsf{continue/abort}), y_2).$$

**Hybrid experiment 1.** The first hybrid experiment proceeds in the same way as the ideal world except, instead of $\mathcal{S}$ simulating the dealer, the true dealer $D$ runs $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}(1^\kappa, x)$ and gives $\mathsf{sh}_i$ to $P_i$.

For the adversary $\mathcal{A}$, let $\text{H1}_\mathcal{A}(1^\kappa, x, f)$ be the joint random variable containing the view of the adversary and the output of the honest party in this hybrid experiment. Concretely,

$$\text{H1}_\mathcal{A}(1^\kappa, x, \ell, f) = ((\mathsf{sh}_1, \mathsf{sh}_1'\ell, y, b^*, \mathsf{continue/abort}), y_2).$$

**Hybrid experiment 2 ($\mathcal{F}^-$-hybrid experiment).** Next, we define a second hybrid experiment, substituting ideal functionality $\mathcal{F}^-$ computing $f_\Sigma$ with one-bit leakage for the ideal functionality computing $f(x)$ with one-bit leakage. This experiment proceeds in the same way as hybrid experiment 1, except that, upon receipt of $\mathsf{sh}_i'$ and $\ell$ from $P_i$, the experiment fowards this value to $\mathcal{F}^-$ (the honest party sends its true share $\mathsf{sh}_{3-i}$) and then returns the output of $\mathcal{F}^-$, namely $y' := f_\Sigma(\mathsf{sh}_1', \mathsf{sh}_2')$ and $b := \ell(\mathsf{sh}_{3-i})$, to $\mathcal{A}$.

For the adversary $\mathcal{A}$, let $\text{H2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, f)$ be the joint random variable containing the view of the adversary and the output of the honest party in this hybrid experiment. Concretely,

$$\text{H2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, \ell, f) = ((\mathsf{sh}_1, \mathsf{sh}_1'\ell, y', b, \mathsf{continue/abort}), y_2).$$

First, we claim the distribution ensembles $\{\text{H1}_{\mathcal{A}}(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\text{IDEAL}_{\mathcal{S}}(1^\kappa, x, f)\}_{\kappa,x,f}$ are indistinguishable.

Suppose, towards a contradiction, that $\mathcal{A}$ is able to distinguish the ensembles with nonnegligible probability. Then we can construct an adversary $\mathcal{A}'$ with nonnegligible advantage in the privacy game for $\Sigma$: $\mathcal{A}'$ chooses two random values $x_0, x_1$ and a bit $i$, which it sends to its game to get $\mathsf{sh}_{b,i}$ of $x_b$ for unknown $b$. It now communicates with $\mathcal{A}$, acting as $\mathcal{S}$ in a run of the ideal experiment with $\hat{x} := x_0$ and $x := x_1$.

At the end of the run, $\mathcal{A}$ guesses whether it participated in an execution of the ideal experiment or hybrid experiment 1. In the former case, $\mathcal{A}'$ returns $b' := 0$; in the latter, it returns $b' := 1$. With nonnegligible probability, the run was an instance in which $\mathcal{A}$ was able to distinguish correctly (without guessing), implying it was able to distinguish between a share of $x_0$ and a share of $x_1$. Thus with nonnegligible probability, $b' = b$ and $\mathcal{A}'$ wins.

This contradicts our assumption that $\Sigma$ is private, so the ensembles must be indistinguishable.

Next, we show the distribution ensembles $\{\text{H1}_{\mathcal{A}}(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\text{H2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, f)\}_{\kappa,x,f}$ are indistinguishable.

Note first that, by definition, $b^* = \ell(\hat{\mathsf{sh}}_2')$ and $b = \ell(\mathsf{sh}_2)$, with $\mathsf{Rec}(\hat{\mathsf{sh}}_1, \hat{\mathsf{sh}}_2') = \mathsf{Rec}(\mathsf{sh}_1, \mathsf{sh}_2) = x$. Since $\hat{\mathsf{sh}}_2'$ and $\mathsf{sh}_2$ are both distributed uniformly at random, $b^*$ and $b$ are identically distributed.

The only other potentially different component in the two ensembles is $y$ (resp. $y'$). Suppose towards a contradiction, that $\mathcal{A}$ is able to distinguish the ensembles with nonnegligible probability. Then we can construct an adversary $\mathcal{A}'$ with nonnegligible advantage in the authenticity game for $\Sigma$. Given $\mathsf{sh}_i$, $\mathcal{A}'$ chooses a random $x$ and receives a share $\mathsf{sh}_i$ of $x$ from the game. It now communicates with $\mathcal{A}$ in a run of hybrid experiment 2, acting as the simulator $\mathcal{S}_2$. When $\mathcal{A}$ sends $\mathsf{sh}_i'$, $\mathcal{A}'$ forwards this value to its game.

At the end of the run, $\mathcal{A}$ guesses which experiment was run. With nonnegligible probability, the run was an instance in which $\mathcal{A}$ was able to distinguish correctly (without guessing), implying $f_\Sigma(\mathsf{sh}_i', \mathsf{sh}_{3-i}) \neq \perp$ and $\mathsf{sh}_i' \neq \mathsf{sh}_i$. So $\mathcal{A}'$ wins with nonnegligible probability. (The other case, in which $f_\Sigma(\mathsf{sh}_i', \mathsf{sh}_{3-i}) = \perp$ and $\mathsf{sh}_i' = \mathsf{sh}_i$, would violate the correctness requirement of $\Sigma$.)

This contradicts our assumption that $\Sigma$ is authenticated, so the ensembles must be indistinguishable.

To conclude, notice that by transitivity, we have computational indistinguishability of $\{\text{IDEAL}_{\mathcal{S}}(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\text{H2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, f)\}_{\kappa,x,f}$, so $\Sigma$ is secure for computing on authenticated data in the $\mathcal{F}^-$-hybrid world. □

Notably, if $\Sigma$ is secure for computing on authenticated data, it also guarantees the integrity of the output $y$, since it guarantees the integrity of $\mathcal{A}$'s input.

## E.5. CoVault security analysis

Finally, we can prove the end-to-end security of the full CoVault protocol. Let $\Sigma_M$ be a secure secret sharing scheme, and for sets $V_1, V_2$, define $\mathsf{Rec}(V_1, V_2)$ as reconstructing each pair of corresponding shares in $V_1, V_2$. That is, $\mathsf{Rec}(V_1, V_2) = \left\{\mathsf{Rec}(\mathsf{sh}_k^1, \mathsf{sh}_k^2)\right\}_{k=1}^{|V_1|}$ (return $\perp$ if $|V_1| \neq |V_2|$). Define the ideal functionality $\mathcal{F}_{\text{asym}}^-$ as a modified version of the ideal functionality of the same name defined in §E.3: $\mathcal{F}_{\text{asym}}^-$ first takes as input two functions $f_1, f_2$. If $f_1 \neq f_2$, the functionality aborts; otherwise, it continues as before, computing $f = f_1 = f_2$. Additionally, when the leakage bit is 1 (without loss of generality), $\mathcal{F}_{\text{asym}}^-$ sends a distinguished message $\mathsf{cheat}$ to the honest party. This captures the probability that a malicious adversary attempting to learn some leakage bit is caught with probability $1/2$ (derived from the DualEx protocol).

**E.5.1. Ideal functionality.** Let $Q$ (a querier) and $DP_1, DP_2$ (two DPs) be potentially malicious parties; an adversary can corrupt at most the set $\{Q, DP_i\}$ of parties for some $i \in \{1, 2\}$.

The parties interact with the ideal functionality $\mathcal{F}_{CV}^-$ as follows. $Q$ sends two (potentially different) functions $f_1, f_2$ to $DP_1, DP_2$, respectively. The (trusted) dealer $D$ send sets $V_1, V_2 \in \mathcal{X}^m$, respectively, to $DP_1, DP_2$. $DP_1$ and $DP_2$ send $f_1', f_2', V_1', V_2'$ to the ideal functionality $\mathcal{F}_{CV}^-$. At this point a corrupted $DP_i$ can additionally send a leakage function $\ell$ to $\mathcal{F}_{CV}^-$.

Upon receiving $f_1', f_2', V_1', V_2', \ell$, $\mathcal{F}_{CV}^-$ computes an output $y$ and evaluates $\ell$ on the honest party's input to get a leakage bit $b$. If $f_1' \neq f_2'$, $y = \perp$; otherwise, it sets $f = f_1'$ (without loss of generality) and computes $y = f_{\Sigma_M}(V_1', V_2')$. The ideal functionality sends $y$ to $Q$ and $b$ to the malicious $DP$. If $b = 1$, it also alerts the honest $DP$ by sending it a distinguished message $\mathsf{cheat}$, in which case that $DP$ aborts.

**E.5.2. Full protocol.** Let $\Sigma_M = (\mathsf{Share}_M, \mathsf{Rec}_M)$ be a secure secret-sharing scheme. Define the following protocol $\Pi$ in the $\mathcal{F}_{\text{asym}}^-$-hybrid model:

1) $Q$ sends two (potentially different) functions $f_1, f_2$ to $DP_1, DP_2$, respectively, who send functions $f_1', f_2'$ to the ideal functionality $\mathcal{F}_{\text{asym}}^-$; an honest $Q$ sends $f_1 = f_2$, and and honest $DP$ sends the function it received

from $Q$. (Notice that leaking the equality of $f'_1, f'_2$ does not reveal additional information to the adversary, since it already knows the two functions.)
2) $D$ sends $V_1, V_2$ to $DP_1, DP_2$, respectively, who send inputs $V'_1, V'_2$ to $\mathcal{F}^-_{\text{asym}}$; an honest *DP* sends the set it received from the dealer. A malicious *DP* can additionally send a one-bit leakage function $\ell$.
3) $\mathcal{F}^-_{\text{asym}}$ computes authenticated shares $y_1, y_2 \in \mathcal{Y}$, where $y_1, y_2 := \mathsf{Share}_M(f_{\Sigma_M}(V'_1, V'_2)))$ and sends $y_i$ to $DP_i$. It also computes $\ell$ evaluated on the honest party's input and sends the leakage bit to the malicious *DP*.
4) If the leakage bit is 1, $\mathcal{F}^-_{\text{asym}}$ alerts the honest *DP* by sending it a distinguished message $\mathsf{cheat}$. Importantly, once cheating is detected, the honest *DP* completes the current execution but never runs any subsequent iterations of the protocol.
5) Each party $DP_i$ sends a value $y'_i$ to $Q$; the honest party sends the value $y_i$ it received from the ideal functionality.
6) $Q$ recovers $y' = \mathsf{Rec}_M(y'_1, y'_2)$.

### E.5.3. Proof of security.

**Theorem 4.** $\Pi$ *securely computes* $\mathcal{F}^-_{CV}$ *in the* $\mathcal{F}^-_{asym}$*-hybrid model against malicious adversaries capable of corrupting at most the set* $\{Q, \text{DP}_i\}$ *for* $i \in \{1, 2\}$.

*Proof.* Let $\mathcal{A}$ be a PPT adversary corrupting parties $Q, DP_i$. To prove the security of $\Pi$ against malicious adversaries in the $\mathcal{F}^-_{\text{asym}}$-hybrid world, we give an adversary $\mathcal{S}$ in the ideal world that interacts with $Q, DP_i$ to simulate an execution of $\Pi$ in the hybrid world.

**Simulator $\mathcal{S}$:** $\mathcal{S}$ has access to the ideal functionality $\mathcal{F}^-_{CV}$. $\mathcal{S}$ works as follows:

- Receive $f_{3-i}$ from $Q$ and $f'_i, V'_i, \ell$ from $DP_i$. Forward $f_{3-i}$ to $DP_{3-i}$.
- Send $f'_i, V'_i, \ell$ to $\mathcal{F}^-_{CV}$; the honest party $DP_{3-i}$ sends $f_{3-i}, V_{3-i}$.
- Receive $y := f_{\Sigma_M}(V'_i, V_{3-i}))$ and some one-bit leakage $b := \ell(V_{3-i})$ from $\mathcal{F}^-_{CV}$.
- Let $y^*_i, y^*_{3-i} \leftarrow \mathsf{Share}_M(y))$. Send $y^*_i, b$ to $DP_i$ and $y^*_{3-i}$ to $Q$.

We now prove indistinguishability of the ideal and $\mathcal{F}^-_{\text{asym}}$-hybrid distribution ensembles.

**Ideal experiment.** This is defined by the interaction of $\mathcal{S}$ with the ideal functionality $\mathcal{F}^-_{CV}$. $\mathcal{A}$ outputs an arbitrary function of its view; the honest party outputs nothing. Let $\text{IDEAL}_{\mathcal{S}}(V, \ell, f)$ be the joint random variable containing the transcript of $\mathcal{S}$ in the ideal world. Concretely,

$$\text{IDEAL}_{\mathcal{S}}(V, \ell, f) = (f_{3-i}, f'_i, V'_i, \ell, y^*_i, b, y^*_{3-i}).$$

**Hybrid experiment.** Let $\text{H}_{\mathcal{A}}(V, \ell, f)$ be the joint random variable containing the view of the adversary $\mathcal{A}$ in the $\mathcal{F}^-_{\text{asym}}$-hybrid world. Concretely,

$$\text{H}_{\mathcal{A}}(V, \ell, f) = (f_{3-i}, f'_i, V'_i, \ell, y_i, b, y_{3-i}).$$

$\mathcal{S}$ perfectly simulates the view of the malicious parties in the hybrid world since the joint distributions of $(y^*_i, y^*_{3-i})$ and $(y_i, y_{3-i})$ are both distributed uniformly at random conditioned on $\mathsf{Rec}_M(y^*_i, y^*_{3-i}) = \mathsf{Rec}_M(y_i, y_{3-i}) = f_{\Sigma_M}(V_1, V_3)$.

Let $p_i$ be the probability the leakage bit is 0 in iteration $i$ (that is, the attacker is not caught). Without loss of generality, we can assume $p_i \geq 1/2$ (if not, the attacker can flip the leakage and learn the same information while lowering its probability of being caught). Then the probability that the attacker is never caught is $p^* = \prod_i p_i$. Define the information learned in iteration $i$ as $I_i = \max\{-\log p_i\} = -\log \max\{p_i\}$. Let $I^* = \sum_i I_i$. Then $I^* \geq -\log p^*$. So $I^* \geq \kappa$ implies $p^* \leq 2^{-\kappa}$, i.e., the probability that the attacker learns at least $\kappa$ bits is at most $2^{-\kappa}$.

Security follows by a hybrid argument and Theorems 1, 2, and 3. □