CoVault: Secure Selective Analytics of Sensitive Data for the Public Good

Roberta De Viti
Max Planck Institute for Software Systems (MPI-SWS)

Isaac Sheff
Max Planck Institute for Software Systems (MPI-SWS)

Noemi Glaeser
Max Planck Institute for Security and Privacy (MPI-SP)
University of Maryland

Baltasar Dinis
Instituto Superior Tecnico (ULisboa)
INESC-ID

Rodrigo Rodrigues
Instituto Superior Tecnico (ULisboa)
INESC-ID

Jonathan Katz
University of Maryland

Bobby Bhattacharjee
University of Maryland

Anwar Hithnawi
ETH Zürich

Deepak Garg
Max Planck Institute for Software Systems (MPI-SWS)

Peter Druschel
Max Planck Institute for Software Systems (MPI-SWS)

Abstract

Many types of analytics on personal data can be made differentially private, thus alleviating concerns about the *privacy* of individuals. However, no platform currently exists that can *technically* prevent *data leakage and misuse* with minimal trust assumptions; as a result, analytics that would be in the public interest are not done in privacy-conscious societies. To bridge this gap, we present *secure selective analytics (SSA)*, where data sources can *a priori* restrict the use of their data to a pre-defined set of privacy-preserving analytics queries performed by a specific group of analysts, and for a limited period. Furthermore, we show that a *scalable* SSA platform can be built in a *strong threat model based on minimal trust.*

Our SSA platform, CoVault, relies on a minimal trust implementation of functional encryption (FE), using a combination of secret sharing, secure multi-party computation (MPC), and trusted execution environments (TEEs). CoVault tolerates the compromise of a subset of TEE implementations as well as side channels. Despite the high cost of MPC, we show that CoVault scales to very large databases using MapReduce-based query parallelization. For example, we show that CoVault can perform queries relevant to epidemic analytics for a country of 80M using about 8000 cores.

1 Introduction

Certain privacy-preserving analytics on high-resolution personal data can be of significant benefit to society. Examples of such data include individuals' education, health, nutrition, activity, mobility, social contacts, income, and finances. This data is already being captured with high resolution and velocity by individuals' smart devices and apps, by online services, and when citizens use offline services that involve digital record-keeping. Large-scale analytics of this data could benefit (i) public health, e.g., by studying epidemics with high spatio-temporal resolution, or new and rare diseases; (ii) sustainability, e.g., by informing transportation, energy, and urban planning; (iii) social welfare, e.g., by uncovering disparities in income and access to education / services; and (iv) economic stability, e.g., by providing insight into markets.

These analytics produce statistical results to which techniques like differential privacy (DP), which formally guarantee individuals' privacy, can be applied. Despite this, liberal societies have foregone such analytics out of concern that sensitive personal data, once collected and aggregated, could be leaked to unauthorized parties or misused. Potential misuse could range from the controversial (e.g., for law enforcement) to the outright undemocratic (e.g., surveillance and discrimination of certain individuals and groups). Hence, the prevailing approach has been to minimize data collection and aggregation to ensure individuals' privacy and prevent data misuse, unauthorized commercial data exploitation, and government overreach. Indeed, existing public sector data is either carefully limited in scope and collected infrequently, e.g., census data, or exists in silos like local government agencies and is unavailable for aggregation and analytics. Data collected by the private sector also tends to be unavailable for analytics in the public interest, partly due to data protection laws (e.g., GDPR, HIPAA) and partly due to commercial interests.

There is a growing realization, however, that this conservative approach conflicts with urgent societal challenges like public health and sustainability, where savings on the order of billion of Euros are thought possible using data-driven innovation in the EU alone [29, 30]. For instance, the European Parliament recently approved the Data Governance

Act (DGA) [3], which aims to promote the availability of protected public-sector data and a trustworthy environment to facilitate its re-use for the public good. We note that a key component of this environment could be a technical platform for performing analytics securely [28, 31].

The key challenge in enabling secure analytics of personal data is an analytics platform that prevents *misuse* and *leakage* of user data using technical means under a suitably strong threat model. In this setting, which we call *secure selective analytics (SSA)*, there are multiple *data sources*, who contribute data for a specific purpose and for a limited period. The untrusted analytics platform aggregates, stores, and performs queries on encrypted data. Finally, there are *analysts* who wish to obtain query results on contributed data.

Key requirements. The SSA platform has the following key requirements: *(1) A strong threat model based on minimal trust:* Given the sensitivity of the raw data and the potential for data leakage and misuse, the platform's security must not rely on trust in any individual party (analyst, software developer, hardware vendor) and the platform must prevent side-channel leaks. *(2) Selective forward consent and confidentiality:* Data sources consent *a priori* to a defined set of analytics queries by a specific group of analysts, for a limited period of time. No other information can be extracted from the data. *(3) Integrity:* Data and query results cannot be altered by the platform or by third parties without detection. *(4) Scalability:* To be of practical use, the platform should be able to perform common analytics queries on large-scale datasets collected from many data sources. *(5) Transparency:* To justify public trust, the platform's design, implementation and configuration should be transparent.

A SSA platform can be useful whenever multiple data sources wish to make their sensitive data available to third-party analysts for a specific purpose. However, this paper focuses on the emerging application area of analytics for the *public good.* Studies indicate that individuals are willing to contribute their data for causes that demonstrably benefit the public good, making this a scenario with high potential impact [5, 59, 73, 94, 95]. Regardless of who benefits from the analytics, however, an SSA platform ensures that data sources retain complete control over who gets to use their data, for what specific purpose, and for how long; it can therefore support analytics in broader scenarios.

Prior work. *Conventional* analytics platforms operate on unencrypted, raw data. Therefore, data sources must trust the platform not to leak raw data, execute queries that data sources have not consented to, or let unauthorized parties execute queries. *Hardware-secured platforms* rely on a Trusted Execution Environment (TEE) [10, 13, 18, 51, 63, 66, 67, 69, 85, 91, 98] to execute queries on otherwise encrypted data within a secure hardware compartment. While more secure than conventional platforms, they are vulnerable to compromise of the TEE or its vendor. Moreover, unless oblivious

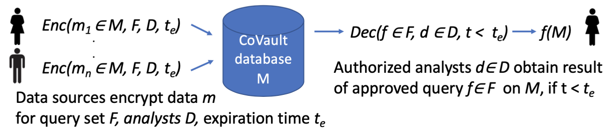
$Enc(m_1 \in M, F, D, t_e)$
$Enc(m_n \in M, F, D, t_e)$
Data sources encrypt data m for query set F, *analysts* D, expiration time t_e
CoVault database M
$Dec(f \in F, d \in D, t < t_e)$
$f(M)$
Authorized analysts $d \in D$ obtain result of approved query $f \in F$ on M, if $t < t_e$

Figure 1. CoVault: Using FE for secure analytics

algorithms are used in the TEEs [105], they are vulnerable to side channels (many such attacks have been reported [75]). The approach does not meet the minimal trust requirement.

Federated analytics systems (FASs) [17, 87–89] leave data in the hands of data sources, thus providing a high degree of confidentiality. However, these systems have many practical limitations: a significant percentage of data sources must be online at any given time, the security protocols used assume substantial bandwidth and computational resources on data sources (which are usually resource-constrained), side-channel leaks are not prevented, and only restricted classes of queries are supported: e.g., with the exception of Mycelium [87], no secure FAS supports queries that walk a private graph and even Mycelium's support for such queries is limited to short bounded paths.

Another approach to secure analytics relies on *homomorphically encrypted (HE) data* [21, 23, 26, 80, 84]. Full HE is very expensive [97]. Partial HE (PHE) restricts query expressiveness significantly, and works efficiently only in weak threat models. For example, the PHE-based systems TimeCrypt [21] and Zeph [22] specifically target the analysis of time-series data with a restricted set of operations (additions but not multiplications), and Zeph operates in a semi-honest threat model. Neither type of HE provides selective forward consent or protection from side-channel leaks by itself.

CoVault. CoVault meets the SSA requirements using a different design based on a form of functional encryption (FE). FE is a generalization of public-key encryption where possessing a decryption key reveals a function f of the encrypted cleartext m, but nothing else about m. Decryption keys are generated by a trusted third party (TTP). FE can be used for secure analytics in an obvious way: Data sources encrypt data using the public key and an analyst wishing to perform query f can ask the TTP for the corresponding decryption key. However, this design places control on decryption in the hands of the TTP, not the data sources, which violates SSA's selective forward consent requirement. To meet this requirement, CoVault adapts FE to move control on decryption from the TTP to the data sources: During encryption, a data source can bind the ciphertext to a set of functions, a set of decryptors (analysts), and a period during which decryption can be performed (see Figure 1).

To meet the strong threat model/minimal trust requirement, CoVault relies on a multi-party FE construction that combines secret sharing, secure multi-party computation (MPC), and multiple TEE implementations. Data is *encrypted* by secret-sharing it to a small set of n parties, which receive and store one share each and jointly implement *decryption* using MPC. MPC protocols require that a subset of the parties be honest (t of n, where t and n depend on the specific MPC protocol). In CoVault, each party executes in a TEE, which ensures honest behavior as long as the TEE is properly attested and remains uncompromised. Moreover, each party uses a *different* TEE implementation to reduce the risk of correlated compromise of parties' TEEs. Unlike designs that rely on a single type of TEE, CoVault can tolerate the compromise, subpoena, and even collusion of up to t TEE types and their vendors. We note that CoVault does not assume physical separation or independent administration of the MPC parties. Instead, trust in CoVault derives from the difficulty for an adversary to simultaneously compromise more than t different TEE implementations. Other players like the datacenter operator can affect only liveness.

To render memory-access side channels ineffective, CoVault uses oblivious protocols. The resulting construction is secure under a threat model that tolerates side channels as well as compromise of any t of n TEE implementations. For integrity, CoVault extends secret-sharing and MPC protocols with message authentication codes (MACs).

To enable scale-out with cores and servers, CoVault relies on oblivious map-reduce, where each mapper and reducer is implemented as an instance of oblivious MPC. For queries that access data based on private attributes, CoVault uses an efficient oblivious data retrieval (ODR) protocol.

To the best of our knowledge, CoVault is the first analytics platform that meets the SSA requirements. CoVault's strong security has a cost, due primarily to the use of oblivious MPC. As we will show in §6, performing epidemic analytics on personal mobility and contact data in a country of 80M using our initial CoVault prototype requires a datacenter with about 8000 cores. We believe this is an acceptable cost for the ability to perform analytics that are of demonstrably high societal value [29, 30] in a manner that respects citizens' privacy. Moreover, rapid advances in MPC protocols promise substantial efficiency increases in the future [37, 46, 79].

Contributions. (a) Motivation and requirements for a secure selective analytics (SSA) platform. (b) The design, implementation, and evaluation of an SSA platform. (c) A multi-party functional encryption construction in a strong threat model that tolerates side channels and compromise of any t of n TEE implementations. (d) An experimental evaluation with epidemic analytics as an example scenario. CoVault's overarching contribution lies in adapting and combining existing cryptographic techniques in a novel way to achieve the SSA requirements (1)–(5).

2 CoVault Overview

Here, we survey relevant building blocks, sketch CoVault's architecture, and provide a roadmap for the rest of the paper.

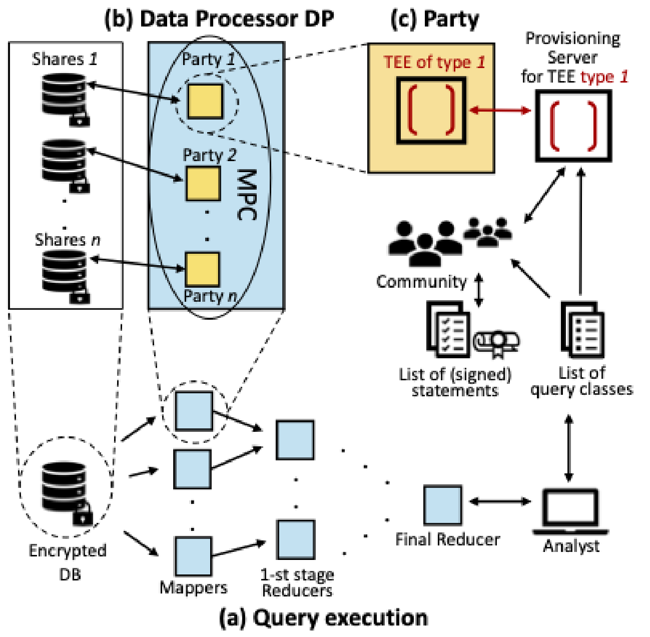

Figure 2. CoVault's Architecture. Queries are executed using oblivious MapReduce on behalf of an authorized analyst (a). Each mapper and reducer is an instance of a Data Processor DP (b), which consists of n parties performing MPC, where each party takes as input one share from the DB. Each MPC party executes in a TEE of a different implementation (c). A provisioning server (PS), one for each TEE type and implemented in a TEE of the same type, attests and provisions with key material all TEEs of a party's implementation. Community witnesses attest the PSs, verify the source code of CoVault and its query classes, and publish signed statements.

2.1 Building blocks used in CoVault

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.

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 [61, 103] or a malicious [101] threat model. In the t-of-n malicious threat model, up to t of the n parties may be malicious, where the value of t varies from $n/3$ to $n-1$ depending on the protocol. Garbled circuits (GCs) are a specific 2PC technique ($n = 2, t = 1$) [83]. GCs are inherently data oblivious as circuits have no control flow.

Trusted execution environments (TEEs) are supported by many recent general-purpose CPUs, e.g., Intel SGX, AMD SEV, and the upcoming Intel TDX and ARM CCA [6, 35, 92, 93]. 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, 35]. 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 address 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. The security of a TEE depends on the integrity of its vendor's certificate chain, proprietary hardware, and firmware.

Functional Encryption (FE) is a generalization of public-key encryption, which enables the holder of a private key to learn a specific function of what the ciphertext encrypts, but nothing else [20]. Iron [49] implements FE using a TEE; this construction is efficient but vulnerable to side channels, and its security relies on the integrity of the (one) TEE's hardware implementation and its vendor.

2.2 CoVault architecture and roadmap

Figure 2 depicts CoVault's architecture. At the heart of CoVault is a multi-party FE-like construction, which we explain by 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 § 5.3 we show how to execute general analytic queries in an efficient and scalable manner. In § 6, we evaluate CoVault in the context of epidemic analytics as an example application scenario. The appendix offers more details on CoVault's community approval process (§ A), oblivious map-reduce technique (§ B), ingress processing (§ C), and epidemic analytics scenario (§ D), with additional evaluation material. Furthermore, the appendix includes our algorithm to estimate end-to-end query latency (§E) and proofs of security of the FE-like construction (§F).

2.3 Threat model

CoVault provides confidentiality and integrity (requirements 2, 3) in the presence of active attacks, including side-channel attacks, without trusting any single software/hardware technology or vendor (requirement 1).

TEEs may be vulnerable to compromise of the vendor's design, implementation, or root of trust, as well as certain physical attacks [75]. Such attacks are in scope of CoVault's threat model. However, it is assumed that the adversary cannot compromise more than any t of n TEE types used in CoVault *simultaneously*. Compromised TEEs are assumed to behave arbitrarily. Moreover, TEEs have been shown to be vulnerable to memory-access side channels [76], which are also in scope. Our prototype implementation uses two parties ($n = 2$) of which any one may be compromised ($t = 1$).

CoVault makes no assumptions about the behavior of data sources, platform operators, analysts, and third parties, with few exceptions: (1) Stakeholders with an interest in CoVault's confidentiality and integrity perform certain actions required of them. Specifically, at least one expert correctly verifies and attests query classes and provisioning servers. Data sources upload their data encrypted with the public key of a query class they consent to and that was approved by an expert they trust. (2) Data sources do not deliberately upload falsified or biased data in order to undermine the quality of analytics results. (Addressing such *data poisoning* attacks is beyond the scope of this paper.)

We make standard assumptions about the strength of cryptographic primitives used in CoVault's design. Furthermore, we assume that data is stored in CoVault for periods short enough that the encryption remains secure despite advances in cryptanalysis and compute power. Securing data longer-term, if required by an analytics application, is beyond the scope of this paper. Denial-of-service attacks are out of scope; they can be addressed with orthogonal techniques.

3 CoVault's FE-like primitive (Basic design)

In this section, we explain CoVault's FE-like construction at a high level, beginning with the API and then incrementally refining strawman designs. § 4 covers details of the construction's realization.

Standard FE allows a party with access to a ciphertext and an appropriate decryption key to compute only 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 a decryptor to evaluate $f(x)$ given the encryption of $x \in X$ and a decryption key sk_f for $f \in \mathcal{F}$ [20]. The decryptor does not learn anything about x other than $f(x)$. In standard FE, a trusted third party decides who gets the decryption key sk_f.

To attain selective forward consent, we use a variant where each instance of FE predefines the set of functions that may be used for decryption (denoted F), the set of authorized decryptors (denoted D), and a time at which all encrypted data expires (denoted t_e). An encryptor should encrypt data using a FE instance only if they are comfortable with the instance's F, D and t_e. We call the triple $[F, D, t_e]$ a *(query) class*, and reflect this FE variation in the $(\mathbf{Setup}, \mathbf{Enc}, \mathbf{Dec})$ API shown in Figure 3. Unlike the standard FE API from the literature, encryption keys are bound to a query class early, there is no *KeyGen* operation or trusted third party, and decryption requires a multi-party computation. Lastly, our FE ensures integrity, which standard FE does not.

We wish to realize this API satisfying two properties:
Confidentiality. Plaintexts remain confidential. Only authorized decryptors (in the set D) may learn authorized functions (from the set F) of the plaintexts if they decrypt before the expiration time t_e. No other entity learns anything.
Integrity. Any modifications to the ciphertext will be detected during decryption.

$MPK[F, D, t_e] \leftarrow \mathbf{Setup}(1^\kappa, F, D, t_e)$
MPC to initialize a new FE instance and produce a public-private keypair $(MPK[F, D, t_e], MSK[F, D, t_e])$ bound to the triple $[F, D, t_e]$. $MPK[F, D, t_e]$ published for encryptors, $MSK[F, D, t_e]$ retained by the n parties. **Inputs:** κ: security parameter, F: set of allowed decryption functions, D: set of authorized decryptors, t_e: expiration time. **Outputs:** n-element public key $(MPK[F, D, t_e])$.

$C \leftarrow \mathbf{Enc}(MPK[F, D, t_e], m)$
Executed locally at a data source to encrypt m for $[F, D, t_e]$. **Inputs:** $MSK[F, D, t_e]$: public key, m: plaintext to encrypt. **Outputs:** C: n shares of the ciphertext.

$f(m) \leftarrow \mathbf{Dec}(f, d, C)$
MPC to produce $f(m)$. If the caller is d, $d \in D$, $f \in F$, and current time $< t_e$, then return $f(m)$, else return $\perp$. **Inputs:** f: function, d: decryptor, C: n shared of the ciphertext. **Outputs:** $f(m)$ or $\perp$.

Figure 3. Our FE API (MPK, MSK, C are n-element vectors)

Next, we describe two strawman designs for our FE that attain these properties under progressively weaker assumptions. The second design matches our threat model (§ 2.3) and forms the basis of our actual FE construction.

3.1 Strawman 1 (S1): *n*-party FE construction

Our first strawman, S1, implements the FE API (Figure 3) by combining secret sharing and n-party MPC. Specifically, n independent parties[1] *jointly* hold shares of the secret key, $MSK[F, D, t_e]$. In S1, confidentiality and integrity rely on at least $n - t$ parties remaining honest.

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 successful 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.

Implementation. Next, we sketch how S1 implements the n-party FE API shown in Figure 3, which can be used after the n parties are initialized.

Setup: Each party executes this function locally, produces a standard asymmetric public-private keypair, keeps its private key locally alongside F, D and t_e, and publishes the public key. MPK and MSK denote n-element vectors (one element for each party/share) of the public and private keys, respectively.

Enc: An encryptor runs this function locally, secret-shares m into n shares, encrypts each share with the corresponding party's public key in the vector $MPK[F, D, t_e]$, and adds message authentication codes (MACs). C is the vector of encrypted and MAC'ed shares, which can be sent to decryptors over any channel.

Dec: The decryptor d calls this function separately on each party, authenticates itself as d, provides the desired function f, and the party's encrypted and MAC'ed share from the vector C. If each party authenticates d, finds that $d \in D$ and $f \in F$, and each party's current clock time is less than t_e, 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 and MAC'ed with the caller's (decryptor's) public key. Otherwise, the result is $\perp$.

Properties. S1 ensures FE semantics because at least $n - t$ parties are assumed to not reveal their shares, and only execute queries in F on behalf of analysts in D, while t_e has not expired. In more detail, S1 provides confidentiality and integrity as long as at least $n - t$ parties are honest. (All others may be arbitrarily malicious). Since S1 does not run parties in TEEs (unlike S2 below), honesty of the $n-t$ parties assumes the absence of malicious intent as well as hardware and software attacks.

3.2 Full construction (S2): S1 + TEE encapsulation

Our full construction, S2, weakens the assumption of *at least $n - t$ honest parties* for the confidentiality and integrity of S1 to one of *at least $n - t$ uncompromised TEE implementations*.

Components. S2 executes each of the n parties inside a TEE of a different implementation. The code of each party is verified to be correct and the corresponding initial measurement of each party's TEE is attested by community experts as part of a review process.

Implementation. During initialization, the systems starts and attests the TEEs for 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 from S1 are executed in TEEs.

Properties. S2 improves S1 by ensuring integrity and confidentiality as long as at least $n - t$ TEEs are uncompromised. The remaining t parties and their underlying TEEs may be arbitrarily compromised. Side-channel attacks against the TEEs are unproductive due to the use of oblivious algorithms.

Discussion: Why TEEs? Without TEEs, designers of any

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

MPC system must reason about the likelihood that parties deliberately deviate or collude, or get compromised or coerced by a third party. Choosing a set of parties so that $> t$ correlated faults are sufficiently unlikely is challenging, and has arguably hindered the adoption of this trust model in practice. With TEEs, on the other hand, community experts can attest the code and initial configuration of each party, and an adversary must compromise TEEs of different types simultaneously. The use of TEEs shifts the root of trust from the trustworthiness of a number of parties to the robustness of an equivalent number of different TEE implementations, which is arguably easier to reason about and discharge in practice. Similar arguments were made in [39].

4 CoVault's FE-like primitive (Details)

CoVault instantiates the S2 construction with two parties, 2PC using garbled circuits, and two TEE implementations ($n = 2$ and $t = 1$). Our choice of 2PC is pragmatic, considering the availability of mature 2PC tools and different TEE implementations at this time. More TEE implementations have been announced, providing an opportunity to pursue an oblivious MPC design with more parties (e.g., using arithmetic circuits) in future work. Such designs may offer better efficiency but involve non-trivial tradeoffs.

In this section, we describe the detailed construction of every function of the FE API (Figure 3). We start with the general setting and system initialization.

General setting. The two parties use different, independent TEE implementations. A party 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 DPs represent a *party* in S2. Each pair of DPs (i.e., one per party) is specific to a pre-determined query class $[F, D, t_e]$ (see below).

Each party's PS holds information on the query classes $[F, D, t_e]$ that have been set up so far. For each triple, this information consists of the measurement hash of the binary code implementing F, the public keys of the authorized decryptors in D, the data expiration time t_e, and the public keys of both DPs that implement that $[F, D, t_e]$.

PSs and DPs can be safely shut down and re-started from their sealed state [33] without re-attestation.

System initialization. Each party instantiates its PS; then, each PS generates its party's private-public keypair. A trusted process separately attests the PSs and signs their public keys. In §5.1, we describe how we implement this trusted process in a decentralized fashion using community approval.

4.1 API call Setup$(1^\kappa, F, D, t_e)$

Each party spawns a fresh DP, and configures it with the class parameters $[F, D, t_e]$ to only compute functions from F for decryptors in the set D, up to time t_e. The DP is remotely attested by its PS and provisioned with cryptographic keys, including the secret key to decrypt its shares of data (in the sense of standard asymmetric cryptography). Each PS stores the class $[F, D, t_e]$ and the public keys of both DPs, and advertises them publicly. The secret and public keys of the DPs are, respectively, the vectors $MSK[F, D, t_e]$ and $MPK[F, D, t_e]$ of the API.

The same trusted process that attested the PSs during system initialization is invoked again to attest the implementation and configuration of each DP.

4.2 API call Enc$(MPK[F, D, t_e], m)$

Encryptors contact the PSs to retrieve information on the available query classes $[F, D, t_e]$, as well as the public keys of the DP pairs that implement them. To encrypt plaintext m for the class $[F, D, t_e]$, an encryptor generates a one-time random pad r and computes two *shares* of m, namely, r and $m \oplus r$, where $\oplus$ is bitwise exclusive-or (xor). It then encrypts each share with the public key of one of the two DPs of $[F, D, t_e]$. This pair of encrypted shares is the output ciphertext denoted C in the API. Secret sharing ensures data confidentiality, while subsequently encrypting the shares prevents data misuse: Only correctly attested TEEs (i.e., attested to implement the specific $[F, D, t_e]$ and thus provisioned with the corresponding decryption keys) can decrypt the shares.

MACs. What we have described so far provides data confidentiality, but not integrity. For integrity, we augment the xor-based secret sharing above with MACs, resulting in a *MAC-then-share* (MtS) scheme. Our MtS works as follows.[2] Let M be any MAC function that provides key- and message-non-malleability and message privacy. An example of such a function M is KMAC256 [64], which our prototype uses. To split m into two shares, the data holder generates a random key k, computes a tag $t \leftarrow \mathsf{M}_k(m)$, and secret-shares k into k_1, k_2 and m into m_1, m_2 using the xor-based scheme above. The two MtS shares of the data are (m_1, k_1, t) and (m_2, k_2, t). Each share is then encrypted with the public key of one of the two DPs (as above); the resulting pair of ciphertexts is the FE ciphertext C.

MtS provides data confidentiality and integrity: The two DPs can *jointly* verify data authenticity by checking that $\mathsf{M}_{k_1 \oplus k_2}(m_1 \oplus m_2) = t$, so neither share can be modified without failing this verification.

4.3 API call Dec(f, d, C)

To decrypt a ciphertext C of a query class, a decryptor d presents the two encrypted shares to the two DPs of the class,

[2] Our design is not tied to this particular MtS scheme. We could also have used other MAC and share schemes from the literature [38, 48].

together with the function f to be applied to the plaintext. The two DPs independently authenticate d and check that f, d and the current time are valid for the query class. If both DPs are satisfied, they locally decrypt their shares, and run a 2PC to reconstruct the plaintext m from the shares, check its MAC, and compute $f(m)$, which is revealed only to the decryptor.

The DPs execute 2PC using garbled circuits (GCs) in the malicious threat model. As the GC protocol, we use DualEx [62], which runs two instances of a standard *semi-honest* GC protocol concurrently on separate core pairs, with the roles of the two parties, conventionally called *generator* and *evaluator,* reversed. DualEx can provably tolerate maliciousness of one party, except for a possible 1-bit leak due to an equality check at the end of DualEx. In the worst case, an adversary can use this leak to disclose any one bit of her choosing in the source data. However, the leak cannot be amplified by repetition and DualEx is orders of magnitude faster than other GC protocols in the malicious model, so we build on it.

Our extension to DualEx. Like most GC protocols, DualEx reveals the computation's result to the two parties. However, we need a protocol that reveals the result to a third entity—the decryptor—but not to the two parties and also adds a MAC that the third entity can verify. For this, we combine DualEx and the MtS scheme of §4.2. We run DualEx to first reconstruct the plaintext m, check its MAC, then compute $f(m)$ and finally compute an MtS on $f(m)$. This yields two shares $s_1 = (f(m)_1, k_1, t)$ and $s_2 = (f(m)_2, k_2, t)$, one of which is returned to each DP by DualEx. Each DP forwards its share to the decryptor over a secure channel. The decryptor reconstructs $f(m)$ as $f(m)_1 \oplus f(m)_2$ and verifies the MAC by checking that $t = \mathsf{M}_{k_1 \oplus k_2}(f(m))$. Neither DP learns $f(m)$ as it has only one share of $f(m)$ and neither DP can change its share without detection by the decryptor.

We note that MtS requires randomness to secret share the payload and the key. In the GC, we obtain this randomness by performing an xor on random bit strings provided by the two DPs. If even one DP is honest, the xor results in a perfectly random string.

Security analysis and proofs. Our assumption that at least 1 TEE implementation is uncompromised (§2.3) and the DP verification process (§4.1) together imply that at least 1 DP of each query class is fully honest. Hence, all GCs are secure. To argue end-to-end security, it only remains to show that our modified DualEx protocol provides confidentiality and integrity, which we do in §F.

5 From FE to the CoVault SSA platform

Next, we describe how we realize the CoVault SSA platform on top of the FE-like primitive from §4.

5.1 Basic operation

Defining a query class. As shown in Figure 1, data sources encrypt their data to a query class $[F, D, t_e]$ so that only functions in F can be executed by analysts in D for a period ending at time t_e. 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. To define a new query class, an interested analyst calls the **Setup** operation of both PSs with the appropriate F, D, t_e.

Contributing data. Data sources choose a query class to which they wish to contribute, execute the **Enc** operation locally using the $MPK[F, D, t_e]$ of the query class to produce the encrypted MtS shares of their data, and send these shares to the DPs associated with the query class. *The DPs store these shares (encrypted and MAC'ed) in a database.* How the database is organized and what if any processing on source data is performed during ingress depends on the database schema and queries in the query class. We sketch an example as part of our epidemic analytics scenario in §6.2. In general, data may be stored with both row- and column-level MACs to enable efficient integrity checks during access.

Querying the database. To run a query f, an analyst invokes **Dec** on a query class that contains f and to which she is authorized. The DPs access (encrypted) data from their database to jointly execute f in 2PC.

Community approval process. CoVault relies on a community approval process to justify trust in its software. In principle, each data source could inspect the source code of the PSs, and the DPs of relevant query classes to see if they correctly implement the query set F and enforce D and t_e, 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. Finally, they would 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 its specification. Separately, community members may remotely attest the PSs and sign the PSs' 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 (DP). Typical analytics queries are of a statistical nature and lend themselves to strong privacy like DP [72]. Query classes with weak privacy should not be approved by community experts, and data sources should not contribute data to them.

5.2 Database access

In general, DB access patterns can leak secrets if the locations accessed depend on secrets read earlier from the DB (data-dependent accesses). A very general solution that hides access patterns is ORAM [52]; however, our experiments showed that even the state-of-the-art Floram [44] is orders of magnitude more expensive than our solution described below. The properties of private information retrieval (PIR) protocols are closer to CoVault's requirements, but still have substantial overhead [34, 36, 78, 99]. Another possible method is to randomly permute secret-shared tables [25, 58, 68], but these techniques either have substantial overhead or they work only in the semi-honest threat model. CoVault instead relies on an oblivious data retrieval (ODR) scheme described below, which is designed to minimize query-time cost at the expense of off-line work. DB accesses that do not reveal secrets, e.g., use a public index (§5.3) or perform a full table scan, need not use ODR.

Oblivious data retrieval (ODR). Our ODR scheme uses a bit of preprocessing inside 2PC followed by pseudorandom table shuffling outside 2PC to hide which DB rows are accessed. The scheme works as follows.

Preprocessing. During data ingress, which runs in 2PC, we preprocess every table that requires ODR access. For every row in the table, we encrypt-then-MAC (EtM) the row, and separately EtM the row's primary key. Secrets used to generate the EtMs are known in 2PC only.

Shuffling. A preprocessed table is shuffled by a pair of DPs (different TEEs), DP_1 and DP_2, *outside 2PC* as follows. First, DP_1 locally shuffles the table by *obliviously sorting* rows, ordered by a keyed cryptographic hash over the primary key EtMs. Oblivious sorting also creates a second layer of encryption over every row. The keys used for hashing and encryption are freshly chosen by DP_1. Next, DP_2 re-shuffles the already shuffled table, by re-sorting the table along a keyed hash over DP_1's hashes using a fresh key. This doubly-shuffled table is stored in the DB indexed by DP_2's hashes.

Row lookup. To fetch a row with a given primary key in 2PC, the position of the row in the stored table is computed by applying the EtM and the two keyed hashes to the primary key. The position is revealed to the two parties, which fetch the row from the shuffled table. Back in 2PC, the encryption layers on the fetched row are peeled off, the MAC of the row's EtM is checked, and the primary key stored within the row is compared with the lookup key.

Properties. The ODR scheme obfuscates the dependence of row locations on primary keys, and protects the integrity and confidentiality of row contents from a malicious party. First, recovering the order of rows in the doubly-shuffled table requires keys of both parties, which only 2PC has in our threat model. Second, neither party learns the content of any row since all data is encrypted by preprocessing using shared keys. Third, any attempt to tamper with a row's data or swap rows is detected by the checks on the fetched row.

Unlike PIR, our ODR scheme does not 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 freshly shuffled tables are readily available to queries (preprocessing happens once per table).

Detecting database roll-back. Untrusted platform operators may roll back the database to an earlier version. In general, known techniques can be used to detect rollback [71, 82], e.g., a secure, persistent, monotonic counter within a TEE or TPM can be tied to the most recent version of the database. In specific cases, such as the epidemic analytics scenario of §6.2, we can instead exploit the fact that the database is append-only and continuously indexed. In this case, rollbacks other than truncation can be detected trivially. To detect truncation, CoVault additionally provides a warning to the querier whenever records within the index range specified in the query are missing.

5.3 Scalable analytics queries

Next, we explain how we perform queries scalably on large tables. (We defer a description of data ingress processing to the appendix, §C.) The unit of querying in CoVault is a standard SQL filter-groupby-aggregate (FGA) query of the following 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 implementation of a FGA query would be a single garbled circuit that takes as input the two shares of the entire dataset 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 may become too large to fit in the memory available on any one machine.

Instead, CoVault implements FGA queries scalably using MapReduce [42]. Mappers and reducers are *separate* GCs and instances of 2PC. There are two challenges in doing this. First, to ensure integrity of data passed from one GC to the next, the first GC must MAC the data (using the MtS scheme), and the next GC must verify the MACs. However, MAC computation is very expensive in GCs, so we would like to minimize it. We address this problem by passing data between GCs – both those that process queries and those at data ingress – in *garbled form*. While this increases the data transmitted or stored by a factor of 256x (128x for the GC coding of each wire and 2x for DualEx), we have empirically found this to be faster than verifying and creating a MAC in each mapper/reducer. Our design reduces in-GC MACs to the minimum possible in the common case: we verify one MAC for every batch of data uploaded by a data source, and create one MAC for every query's output. The only exceptions are database fetches using ODR; these fetches require additional MAC and hash operations as explained in § 5.2.

Second, query execution must be oblivious, i.e., memory accesses and the volume of network communication must not reveal information about private data. To ensure that the volume of network communication betweeen mappers and reducers does not reveal the size of (intermediate) computation values, we pad all traffic between nodes to its maximal expected size, denoted d. Memory accesses are inherently oblivious as GCs lack control flow. However, this also means query processing must use *oblivious algorithms*, which do not require data-dependent control flow. CoVault's mappers and reducers rely on oblivious algorithms for sorting a list, merging two sorted lists and compacting a list (removing marked records from the list): bitonic sort and bitonic merge [15], and a butterfly circuit for compaction [53, §3]. Each of these oblivious algorithms is $O(\log(n))$ slower than the fastest non-oblivious algorithm for the same task, where n is the size of the input.

Public index. We can speed up queries using a public attribute of a table by creating a *public index* of the attributes and avoid filtering in 2PC. Public indexes can also speed up queries that *join* data on a public attribute (an example of this join optimization is in § D.2). Otherwise, joins can be very expensive when done obliviously [105], even more so in 2PC. In § 6.2, we show queries that exploit public indexes.

6 Evaluation

Next, we present results of an experimental evaluation to address the following high-level questions: What is the cost of basic query primitives and the random shuffling required for data-dependent queries? How does query latency scale with the number of cores for a set of realistic epidemic analytics queries? We describe our prototype and experimental setup and then define an epidemic analytics scenario as an example application. Next, we present result of microbenchmarks, followed by results for epidemic analytics at scale. We close with a discussion of results.

6.1 Prototype and experimental setup

We implemented CoVault using EMP-toolkit [100], a library that compiles and executes C/C++ programs in garbled circuits-based 2PC, using recent optimizations such as Half Gates [104] and FreeXOR [65]. We use bitonic sort and bitonic merge from the EMP library, and implement our own primitives for compaction, linear scan, and the MtS of § 4.2 (based on KMAC256, using a pre-existing circuit for Keccak). 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 § 5.3. We implemented the two-party shuffle operation from § 5.2 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 CPUs, respectively. We use 7 machines with Intel® Xeon® Gold 6244 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.[3] 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. We believe the resulting performance of 2PC is conservative, because it is limited by the 2.8GHz AMD CPUs, which do use TEEs and are slow compared to the fastest AMD and Intel CPUs available (3.5GHz).[4] We use at most 8 cores on each of the AMD and Intel machines (cores are used in pairs).

6.2 Example scenario: Epidemic analytics

We evaluate CoVault in the context of epidemic analytics, but note that the latter is only an example scenario: CoVault's query processing overhead is largely independent of the data semantics, which impacts at the most the choice of the upper bound on the possible output sizes (see § 5.3). Hence, 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

[3]Such network connectivity is common in datacenters. Because trust in CoVault relies on different TEE implementations, there is no need for physical separation or geo-distribution of parties.

[4]While the relative performance of Intel TDX and AMD SEV-SNP is unknown, we think it will very likely be comparable: TDX has virtually identical capabilities (including hardware memory encryption) and was evidently designed to compete with SEV-SNP, which is already used in Cloud services by Google and Azure [12, 27]. (SGX was not designed for the server/Cloud market and does not support encapsulated VMs.)

Figure 4. Schema of views for epidemic analytics.

T_E	*space-time-region*	eid	did1	did2	…		
T_P	*epoch*	did1, time	did2	duration	prev	next	…

Figure 5. Queries used in the evaluation. The selections on the public attributes `space-time-region` and `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 < epoch < end)
SELECT COUNT(DISTINCT(did2)) FROM TT
WHERE did1 ∈ A AND did2 ∈ B
 AND start < time < end
```

locations and pairwise Bluetooth radio encounters reported by smartphones [70, 96]. Real data of this sort would be sensitive, and we assume that users would voluntarily provide such data for epidemic analytics only to a SSA platform.

The ingress processing for the epidemic analytics scenario is detailed in § C, § D.2. Ingress results in two materialized views, T_E and T_P, whose schemas are shown in Figure 4. 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. All regions are padded to the same length to obfuscate the actual number of records in a region.

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 an *epoch*. T_P is used by data-dependent queries, so this view is shuffled (§ 5.2).

Our evaluation uses the queries q1–q3 shown in Figure 5, developed in consultation with an epidemiologist. Queries

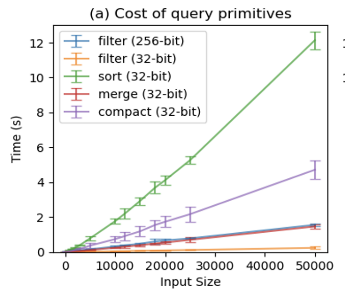

Figure 6. Processing time vs. workload size

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 § D.4, extends q3 to indirect encounters between `A` and `B` via a third device.

6.3 Microbenchmarks

We first report the costs of basic query primitives (§ 5.3), mappers and reducers for a generic FGA query (§ 5.3), and random shuffling (§ 5.2). These primitive costs can be used to estimate the total cost of arbitrary FGA queries, beyond the specific epidemic analytics queries we evaluate in § 6.4, assuming sufficient internal network bandwidth, which is normally available in a datacenter.

All experiments reported here run on a single pair of cores, one Intel and one AMD. We report the cost of only one of the two executions of DualEx since the two executions are completely independent in CoVault, execute in parallel, and synchronize only once at the end of a query for DualEx's equality check (performed 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.

Query primitives. Figure 6a shows the time taken to execute the basic oblivious query primitives from § 5.3—linear scans on 32-bit and 256-bit records, sort, sorted merge, and compact on 32-bit records—as a function of the list length. Each reported number is an average of 100 measurements, with std. dev. shown as error bars.

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. As expected, 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, as explained below.

Map and reduce. Figure 6b shows the average time taken to execute, on a single core pair, a typical FGA query. (The specific operations are those of query q2 in Figure 5.) The map operation linearly scans input records and projects a column of records whose attributes match a given value. The 1st-stage reducer concatenates the outputs of two mappers, sorts the concatenated output by the grouping key of the query, and computes aggregates for all groups. The subsequent-stage reducers do the same, except that their inputs are already sorted, so they can merge the inputs using a more efficient circuit.

The theoretical complexity of a mapper, 1st-stage reducer, and subsequent-stage reducer is $O(c)$, $O(c(\log(c))^2)$ and $O(d\log(d))$, respectively, where c is the input size and d is the maximal padded size between reducers. The 1st-stage reduction is more expensive due to the additional sort operation. The empirical results are as expected: The mapper is linear in the input size, while the reducers are slightly super-linear, with the 1st-stage reduction more expensive.

Shuffling. Each shuffle requires two sequential oblivious sorts in TEEs of different types, outside 2PC. Shuffling a view of 600M records takes about 56min on a single core pair. Shuffling one-tenth the number, 60M records, takes 4min. The variances are negligible. The scaling is super-linear because oblivious sort (even outside 2PC) runs in $O(n(\log(n))^2)$ steps.

A shuffle is used for only one query, 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 of 600M records—a conservative upper bound on the encounters generated by a country of 80M people in 1h. Therefore, for a country of this size, we can prepare shuffles for q queries using $q/2$ pairs of cores continuously.

Bandwidth. Garbled circuits stream 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 ~2.9Gbps. The bandwidth from the evaluator to the generator is negligible in comparison. With DualEx, the average bandwidth would be 2.9Gbps in each direction. Thus, 8 active cores on each machine need a bandwidth of 11.6Gbps in each direction, which our 2x25Gbps links can easily support.

Storage overhead, garbled storage. Storing data in CoVault's DB in garbled form has a storage overhead of 512x relative to an insecure base line: 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 a conservative calculation, a country

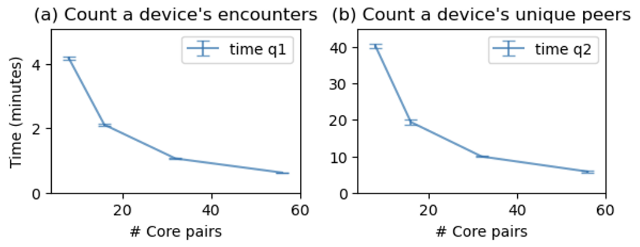

Figure 7. Query latency vs. total number of core pairs (including the parallel DualEx execution) for the FGA query of (q1) and (q2) from Figure 5. The measurements are the average of 20 runs, with std. dev. shown as error bars.

with population 80M generating 11.85B encounters per day (encs/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. (Raw data can normally be discarded after a short time; e.g., after 14 days in the epidemic scenario.) This storage overhead can be be reduced to about 2x when storing data in non-garbled form (two shares per value, plus space for one MAC per column and space-time region or epoch, see §5.3). However, this comes at the expense of having to verify MACs during each query, which is expensive (see below).

6.4 Epidemic analytics: Query latency

We measured end-to-end query latency, as a function of input size and core count, for the queries q1–q3 of Figure 5.

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 $\mathbb{R}$, using the public index of T_E, which significantly reduces the size of the input table and the query latency. In both queries, we iterate over devices in the given set $\mathbb{A}$. For each device a in $\mathbb{A}$ we issue a FGA query. For q1, this FGA query counts the number of encounters of device a in $\mathbb{R}$. For q2, this FGA query counts the number of unique devices that device a met. Note that FGA queries for all devices in $\mathbb{A}$ can be done in parallel, independently.

Figure 7a shows the latency of q1 against the number of core pairs. 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 encs/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, showing near-linear scaling with cores.

Figure 7b shows the latency of q2 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 §5.3, we need to specify an upper bound on the size of each reducer's output. Here, we fix it to 500 (i.e., we assume that a person will encounter at most 500 unique devices in 14 days). Query latency varies almost inversely with the number of cores, but is higher than for q1 since first-stage reducers do more work in q2.

Cost breakdown for q2. For q2, we measured the latency contributions of different design elements. A single (semi-honest) GC execution takes 92.2% of the reported latency, adding the parallel DualEx execution on a separate pair of cores plus the DualEx equality check adds 6.2% to the latency, and executing inside a TEE adds a mere 1.6%. The cost of adding DP noise is negligible (on the order of *ms*). For reference, adding a per-column MAC verification, which would be required if data was not stored in garbled form, would add a more than four-fold overhead to the reported latencies.

Scaling to many cores. 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 (e.g., 46 16-CPU machines). For reference, if data were stored in non-garbled form, the number of cores required to execute this query in the same time would increase by 124%. The details of our extrapolation are in §E.

Query q3. Query q3 runs on T_P. A naive implementation of q3 in 2PC requires a linear scan of all encounter records during the period of interest (`[start,end]`) to find those between two peers in sets `A` and `B`. Based on our earlier estimate, for a table of 80M people over a 14-day period, this would require loading and filtering 165.9B records *in 2PC*—a formidable task. To make this query efficient, we rely on T_P's private index on one of the encounter peers, and use our shuffled ODR to securely load only those encounters that involved a peer from set `B`. For `A` and `B` of size 10 each, this reduces the number of records read and processed in 2PC by 5 orders of magnitude to ~100K. The total query latency on 2 pairs of cores in our setup with these parameters is 2.24 hours. We refer to §D.4 for additional details of the view T_P and the query implementation.

6.5 Discussion

Query execution cost. Query execution cost is dominated by MPC, i.e., garbled circuits in our prototype. As such, CoVault will be able to benefit from future advances in MPC protocols, e.g., honest-majority oblivious MPC protocols with more than two parties. For data-dependent queries, the cost of shuffling is also significant. Non-MPC computation, TEEs, database access, network communication have insignificant overhead, given the high-speed connectivity available in datacenters. At the level of map-reduce, the mappers and 1st-stage reducers take most of the time, because they ingest all data used in a query and bottleneck on available cores.

Scalability. CoVault's combination of garbled circuits and map-reduce scales well: CoVault can analyze large datasets with reasonable latency using proportionally more cores. In the epidemic analytics scenario, for a 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 §D.3), and running an epidemic analytics query like q2 on 14 days of such records (165.9B records) within 24h requires an additional 2,320 core pairs engaged for the 24h of the query's execution. Looked at differently, a core pair is required for roughly every 20,000 citizens at this scale. Although high in absolute number, these core requirements correspond to mid-sized datacenters operated by individual organizations today, and we believe that such an investment is justifiable when the analytics has a sufficiently high socio-economic benefit, which is the case for epidemic analytics, for instance.

Query expressiveness. In principle, CoVault can execute any FGA query. However, oblivious joins on columns without a public index are expensive. This cost can be amortized by creating and storing views containing all joins relevant to a query class during ingress. Another optimization, which we use for the query q3 (§6.4), is to filter records before joins.

7 Related work

We discussed prior work on hardware-secured, federated, and homomorphic encryption-based analytics in §1. Here, we discuss other related work. We note that CoVault is the first analytics platform that provides the five SSA properties from §1 simultaneously: a strong threat model that divides trust and tolerates side channels, confidentiality with selective forward consent, integrity, scalability and transparency.

Encrypted databases. Early secure DBs like CryptDB [84] use weak encryption like deterministic or order-preserving encryption, which were shown to be insecure [19, 56]. Blind Seer [81] uses strong encryption and 2PC to traverse a specialized index, but leaks information about its search tree traversal and supports only a limited set of search queries. Neither system protects against side-channels leaks.

Data-oblivious analytics. M^{2}R [43], Opaque [105], its extension MC^2 [32], OCQ [41], StealthDB [55] and Ohrimenko *et al.* [76, 77] are secure analytics platforms that run encapsulated in a (single) TEE but, additionally, use oblivious algorithms to mitigate various side-channel leaks. Cipherbase's oblivious extension [11] proposes optimized, oblivious implementations of query operators but, to the best of our knowledge, this extension was not implemented. ObliDB [47] extends Opaque and Cipherbase with optimized operators that do not need to scan the whole table for every query. CoVault's threat model is stronger than these systems' as it divides trust among several TEE implementation.

Analytics using secret sharing. Obscure [57], Cryptϵ [26], and GraphSC [74] use secret sharing to distribute trust over multiple parties, and MPC or partially homomorphic encryption to process queries. Waldo [40] uses secret sharing and 3PC, relying on trust in 2 out of 3 servers, but focuses on outsourced analytics of time-series data from a *single* source. SMCQL [16] uses MPC to eliminate expressiveness limitations that are common in federated analytics systems. Unlike CoVault, these systems do not encapsulate MPC in TEEs and, therefore, rely on non-technical means to prevent correlated attacks. They also do not provide selective forward consent.

Combining MPC and TEEs. Using TEEs to bootstrap decentralized trust has been proposed in a recent workshop paper [39], but not in the context of an analytics platform. Prior work combining MPC and TEEs addresses other problems (e.g., smart contracts [86]), does not use multiple TEE implementations or does not address side channels. A recent paper [102] describes a database combining MPC and TEEs. However, the system assumes trusted clients (analysts), does not support selective forward consent and was not shown to scale to large databases. No prior work we know of combines MPC and TEEs in a way that meets the SSA requirements.

Functional encryption. We covered FE in §2 and §3. Here, we discuss additional work. *Collusion-resistant* FE (e.g., [8]) provides the additional property that the adversary cannot learn anything new by combining the decryption keys of different functions. Although the properties we listed do not capture this, our FE-like implementation has this property. The scheme in [8] was not implemented and its practical efficiency is unclear. *Worry-free encryption* [90] is a public-private-key FE variant where the recipient can decrypt the function of the plaintext only if the recipient satisfies some policy. However, the encryption is to a specific public key. This is a misfit for SSA, where the specific analyst who will make the query is not necessarily known when data is encrypted (this limitation is also reported in [90], §1.1).

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] Council approves Data Governance Act, 2022. https://tinyurl.com/consilium-europa-dga.

[4] S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption: New perspectives and lower bounds. Cryptology ePrint Archive, Paper 2012/468, 2012. https://eprint.iacr.org/2012/468.

[5] M. Aitken, J. de St Jorre, C. Pagliari, R. Jepson, and S. Cunningham-Burley. Public responses to the sharing and linkage of health data for research purposes: a systematic review and thematic synthesis of qualitative studies. *BMC medical ethics*, 17(1):1–24, 2016.

[6] 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.

[7] P. Ananth, A. Jain, Z. Jin, and G. Malavolta. Pre-constrained encryption. In M. Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[8] P. Ananth and V. Vaikuntanathan. Optimal bounded-collusion secure functional encryption. Cryptology ePrint Archive, Paper 2019/314, 2019. https://eprint.iacr.org/2019/314.

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

[10] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal Security With Cipherbase. In *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*, January 2013.

[11] A. Arasu and R. Kaushik. Oblivious Query Processing. *CoRR*, abs/1312.4012, 2013.

[12] Azure. Azure and AMD announce landmark in confidential computing evolution . https://azure.microsoft.com/en-us/blog/azure-and-amd-enable-lift-and-shift-confidential-computing/.

[13] S. Bajaj and R. 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] G. Barthe, R. De Viti, P. Druschel, D. Garg, M. Gomez-Rodriguez, P. Ingo, M. Lentz, A. Mehta, and B. 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] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. SMCQL: Secure Querying for Federated Databases. *Proceedings of the VLDB Endowment*, 10(6):673–684, February 2017.

[17] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers. ShrinkWrap: Efficient SQL Query Processing in Differentially Private Data Federations. *Proc. VLDB Endow.*, 12(3):307–320, 2018.

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

[19] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov. The Tao of Inference in Privacy-Protected Databases. *Proceedings of the VLDB Endowment*, 11(11):1715–1728, July 2018.

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

[21] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. 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.

[22] L. Burkhalter, N. Küchler, A. Viand, H. Shafagh, and A. Hithnawi. Zeph: Cryptographic enforcement of end-to-end data privacy. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*, pages 387–404, 2021.

[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] D. Cerdeira, N. 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] M. Chase, E. Ghosh, and O. Poburinnaya. Secret shared shuffle. Cryptology ePrint Archive, Paper 2019/1340, 2019. https://eprint.iacr.org/2019/1340.

[26] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin

Machanavajjhala, and Somesh Jha. Cryptϵ: Crypto-assisted differential privacy on untrusted servers. In *Proc. SIGMOD*. ACM, 2020.

[27] Google Cloud. Security through collaboration: Building a more secure future with Confidential Computing . https://cloud.google.com/blog/products/identity-security/google-amd-partner-to-build-a-more-secure-future-with-confidential-computing.

[28] European Commission. Data Governance Act explained. https://digital-strategy.ec.europa.eu/en/policies/data-governance-act-explained.

[29] European Commission. European data governance act. https://digital-strategy.ec.europa.eu/en/policies/data-governance-act.

[30] European Commission. European data strategy. https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-data-strategy_en.

[31] European Commission. Public Consultation on the Data Act: Summary Report . https://digital-strategy.ec.europa.eu/en/library/public-consultation-data-act-summary-report.

[32] MC^2 Contributors. MC^2: A Platform for Secure Analytics and Machine Learning. https://mc2-project.github.io/index.html.

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

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

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

[36] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal Service-Providers for Private Information Retrieval. *J. Cryptology*, 14:37–74, 2001.

[37] A. Dalskov, D. Escudero, and M. Keller. Fantastic Four:{Honest-Majority} {Four-Party} Secure Computation With Malicious Security. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2183–2200, 2021.

[38] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[39] E. Dauterman, V. Fang, N. Crooks, and R. A. Popa. Reflections on trusting distributed trust. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 38–45, 2022.

[40] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468, 2022.

[41] A. Dave, C. Leung, R. A. Popa, J. E. Gonzalez, and I. 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.

[42] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. 2004.

[43] T. T. A. Dinh, P. Saxena, E. Chang, B. C. Ooi, and C. 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.

[44] J. Doerner and A. Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 523–535, New York, NY, USA, 2017. Association for Computing Machinery.

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

[46] D. Escudero, V. Goyal, A. Polychroniadou, and Y. Song. TurboPack: Honest Majority MPC with Constant Online Communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 951–964, 2022.

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

[48] Saba Eskandarian and Dan Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. *Cryptology ePrint Archive*, 2021.

[49] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. Iron: functional encryption using intel sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 765–782, 2017.

[50] B. A. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. Iron: Functional encryption using intel sgx. Cryptology ePrint Archive, Report 2016/1071, 2016. https://ia.cr/2016/1071.

[51] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A. R. Sadeghi. HardIDX: Practical and secure index with SGX. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 386–408. Springer, 2017.

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

[53] M. 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.

[54] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. Cryptology ePrint Archive, Paper 2012/521, 2012. https://eprint.iacr.org/2012/521.

[55] A. Gribov, D. Vinayagamurthy, and S. Gorbunov. StealthDB: a scalable encrypted database with full SQL query support. *CoRR*, abs/1711.02279, 2017.

[56] P. Grubbs, T. Ristenpart, and V. 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.

[57] P. Gupta, Y. Li, S. Mehrotra, N. Panwar, S. Sharma, and S. Almanee. Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. *Proceedings of the VLDB Endowment*, 12(9):1030–1043, May 2019.

[58] William L. Holland, Olga Ohrimenko, and Anthony Wirth. Efficient oblivious permutation via the waksman network. In *Proc. ASIA CCS*. ACM, 2022.

[59] N. Howe, E. Giles, D. Newbury-Birch, and E. McColl. Systematic review of participants' attitudes towards data sharing: a thematic synthesis. *Journal of health services research & policy*, 23(2):123–133, 2018.

[60] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols?

[61] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, number 1, pages 331–335, 2011.

[62] Y. Huang, J. Katz, and D. 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.

[63] T. Hun, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 533–549, Savannah, GA, November 2016. USENIX Association.

[64] J. Kelsey, S. Chang, and R. Perlner. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, 2016. NIST Special Publication 800-185. https://doi.org/10.6028/NIST.SP.800-185.

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

[66] OASIS labs. A better way to Contact Trace, Part I. https://medium.

com/oasislabs/a-better-way-to-contact-trace-7beb12889017.

[67] 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.

[68] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In *Proc. ISC*. Springer, 2011.

[69] U. Maheshwari, R. Vingralek, and W. 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.

[70] J. Manweiler, R. Scudellari, and L. 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.

[71] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun. ROTE: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1289–1306, Vancouver, BC, August 2017. USENIX Association.

[72] F. McSherry. Privacy integrated queries. *Communications of the ACM*, 53:89–97, September 2010.

[73] M. M. Mello, V. Lieou, and S. N. Goodman. Clinical trial participants' views of the risks and benefits of data sharing. *New England journal of medicine*, 378(23):2202–2211, 2018.

[74] 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.

[75] A. Nilsson, P. N. Bideh, and J. Brorsson. A Survey of Published Attacks on Intel SGX, 2020.

[76] O. Ohrimenko, M. Costa, C. Fournet, C. Gkantsidis, M. Kohlweiss, and D. Sharma. Observing and Preventing Leakage in MapReduce. In I. Ray, N. Li, and C. 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.

[77] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. 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.

[78] Femi G. Olumofin and Ian Goldberg. Privacy-Preserving Queries over Relational Databases. In M. J. Atallah and N. 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.

[79] E. Orsini. Efficient, actively secure mpc with a dishonest majority: a survey. In *International Workshop on the Arithmetic of Finite Fields*, pages 42–71. Springer, 2021.

[80] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. 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.

[81] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. 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.

[82] B. Parno, J. Lorch, J. Douceur, J. Mickens, and J.M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2011.

[83] B. Pinkas, T. Schneider, N. P. Smart, and S. 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.

[84] R. A. Popa, C. MS. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[85] C. Priebe, K. Vaswani, and M. 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.

[86] Q. Ren, Y. Wu, Y. Li, H. Liu, H. Lei, L. Wang, and B. Chen. Tenet: Towards self-sovereign and fair multi-party computation ecology empowered by decentralized tee network. *arXiv preprint arXiv:2202.10206*, 2022.

[87] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen. *Mycelium: Large-Scale Distributed Graph Queries with Differential Privacy*, page 327–343. Association for Computing Machinery, New York, NY, USA, 2021.

[88] E. Roth, D. Noble, B. H. Falk, and A. 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.

[89] E. Roth, H. Zhang, A. Haeberlen, and B. 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.

[90] A. Sahai and H. Seyalioglu. Worry-free encryption: Functional encryption with public keys. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 463–472, New York, NY, USA, 2010. Association for Computing Machinery.

[91] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. 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.

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

[93] 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.

[94] K. Spencer, C. Sanders, E. A. Whitley, D. Lund, J. Kaye, W. G. Dixon, et al. Patient perspectives on sharing anonymized personal health data using a digital system for dynamic consent and research feedback: a qualitative study. *Journal of medical Internet research*, 18(4):e5011, 2016.

[95] J. Stockdale, J. Cassell, and E. Ford. "giving something back": a systematic review and ethical enquiry into public views on the use of patient data for research in the united kingdom and the republic of ireland. *Wellcome open research*, 3, 2018.

[96] L. Tsai, R. De Viti, M. Lentz, S. Saroiu, B. Bhattacharjee, and P. 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.

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

[98] R. 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.

[99] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical Private Queries on Public Data. In A. Akella and J. 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.

[100] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016. Accessed: 2020-05-27.

[101] X. Wang, S. Ranellucci, and J. Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. Cryptology ePrint Archive, Report 2017/030, 2017. https://eprint.iacr.org/2017/030.

[102] P. Wu, J. Ning, J. Shen, H. Wang, and E. Chang. Hybrid trust multiparty computation with trusted execution environment. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

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

[104] S. Zahur, M. Rosulek, and D. 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.

[105] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. 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.

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.

B Further details on scalable analytics queries

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

FGA queries. A FGA 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 one machine. To exploit core parallelism and to make circuit size manageable, CoVault relies on the observation that FGA queries can be implemented using MapReduce [42]. We first explain how this works in general (without 2PC) and then explain how CoVault does this in 2PC.

Background: FGA queries with MapReduce. Suppose we have m cores available. 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 MapReduce 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 MapReduce 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 [53, §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 as discard 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 records marked as discard 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 use $c = 10k$.

Data-dependent FGA queries. If a query accesses a table via a private index or if the data rows accessed by a FGA query *depend* on the output of an earlier query, information about database content may leak via the DB access pattern. To avoid such leaks, data-dependent FGA queries use previously shuffled tables (§5.2). Also, the number of records read must not depend on previous query results; to this end, CoVault adds dummy record reads.

C Ingress processing 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 region. If the (spatio-)time attribute reveals no sensitive information, we can exploit data locality: grouping and storing together data according to *public* (spatio-)temporal information speeds up the queries fetching a whole group for processing. To exploit data locality, CoVault creates a materialized view(s) indexed by a public attribute(s), e.g., coarse-grained (space-)time information. Thus, CoVault is a read-only platform but supports DB appends in order of public attributes: 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 may perform pre-processing operations in 2PC; for instance, the IPs can buffer different batches and run group or sort operations (in 2PC) in order to produce or append to materialized views. The actual operations and materialized views are application-specific; in §D, 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 during upload.

Storing data in garbled form. As discussed in § 5.3, 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 in 2PC, and for the DPs to verify them. (However, recall that IPs need to *verify* in 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 class only.

D Further details on the epidemic scenario

D.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 [70, 96], 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 8. There is a separate view for every query class. The views include

View	Record fields							
T_E	*s.-t.-region*	eid	did1	did2	t	dur	aux	validity
T_S	*s.-t.-region*	did1	aux	period of contagion				
T_P	*epoch*	did1, t	did2	dur	prev	next	aux	validity

Figure 8. Materialized views used for epidemic analytics. The first column is a public index. (*s.t.* = *space-time*).

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 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 §D.2). Additionally, the records may report *t*, a fine-grained temporal information indicating when an encounter begins, *dur*, the encounter duration, and *aux*, any additional information not currently used (e.g., fine-grained location or signal strength). Finally, note that the records in T_P are organized by device trajectories: these records are indexed by *did*1 and *t*, and each entry contains the indexes of the previous (*prev*) and next (*next*) encounters of *did*1. This view is used by queries that search trajectories in an encounter graph (q3 in Figure 5 and q4 in Figure 10).

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. 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: prior to run the query in 2PC, the DPs can locally fetch all records whose coarse-grained index falls within the space-time region in input.

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 it contains. (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 § 5.3, space-time views can be used only if the queries do not require any secret-dependent data access. Otherwise, we use shuffled encounter views, which support our ODR scheme (§ 5.2). These views allow running a sequence of FGA 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 conservative coarse-grained *time* information, which we call *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 in 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.

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.

D.2 Ingress Processing

Next, we discuss CoVault's ingress processing (§ C) specific to epidemic analytics. As mentioned in § D.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. Then, the data source secret-shares and MACs the batches following the protocol of § 4.2, 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 pair runs a 2PC, which:

1. consumes all per-device batches for the space-time region
2. reconstructs the batches from the shares
3. verifies the per-batch MACs
4. merges the (sorted) batches into a space-time region buffer using oblivious sorted merge [60]
5. 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 confirm encounters within 2PC. 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 § C) to the appropriate tables and materialized views, performing random shuffling when needed.

D.3 Evaluation of Ingress Processing

Ingress processing 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 completely independently in parallel (the tables/views created by the IPs are stored in their in-circuit, garbled form).

In Figure 9, the two highest, nearly overlapping lines 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 encounters/batch or 100 encounters/batch) on a pair of cores (i.e., a pair of IPs). The IPs performs all the (1–5) steps in § D.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 3rd line, which shows the cost without it). 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, 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 9 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 data rate generated by a given administrative entity (e.g., country, city). Thus, we run a 5h experiment where *m* core pairs generate space-time regions of 100 records each from uploads in batches of size 20. We measured the

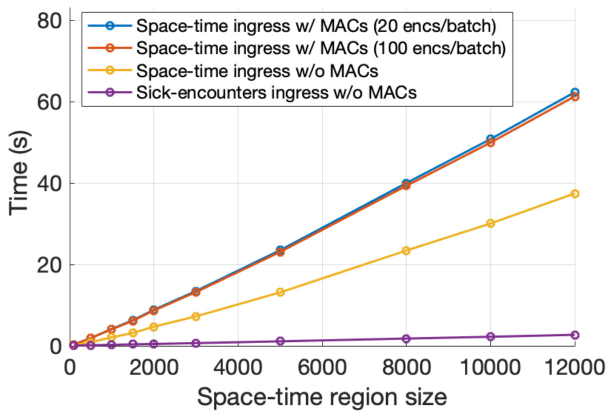

Figure 9. Cost of ingress processing on a single space-time region as a function of the region size.

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, conservatively assuming 200 encounters/person/day in urban areas and 100 encounters/person/day in rural areas, we estimate that, every 24h:

- a mid-sized country with population 80M would generate 11.85B encounters;
- a big metropolitan area with population 8M would generate 1.6B encounters;
- a urban city with population 3M would generate 600M encounters;
- a urban town with population 200k would generate 40M encounters.

Extrapolating from the linear scaling above, we estimate that ingress processing needs a total of 1660, 226, 86, and 6 core pairs, respectively. 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.

D.4 Evaluation of data-dependent queries

Details of the implementation of 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 *epochs*, from the epoch that contains `end` to the epoch that contains `start`. This iteration over epochs is done outside 2PC since epochs are public. The data for each 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 epoch to their earliest encounter, by following encounter pointers. For this traversal, 2PC makes data-dependent queries to the epoch's records in `TT` but since the records in each epoch are randomly shuffled per our ODR scheme, this does not reveal any secrets. To ensure that no secrets leak via the *number* of lookups in an epoch, we use a fixed, conservative number of lookups per device (in `B`) per 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 membership in 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 $e = 336$ epochs (corresponding to a period of 14 days with 1h 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 the ODR scheme improves the latency of q3 significantly by enabling secure data-dependent accesses. Without ODR, we would have to scan *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 11.85B·14 = 165.9B 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 over 6 orders of magnitude fewer records fetched and processed in 2PC.

An additional query: q4. Here, we describe another data dependent query q4 (Figure 10), which we also evaluate for latency.

(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 < 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
```

Figure 10. Query q4. The selection on the public attribute epoch is done outside 2PC using T_P's public index.

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.

We implemented and evaluated the latency of q4 in the same setting as q3 (§6.4). 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 A ∪ 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 A ∪ B has negligible cost in comparison. Assuming $e = 336$ epochs, $n = 30$ encounters/person/h, $|\mathtt{A}| = 10$ and $|\mathtt{B}| = 10$ (the same values that we assumed for q3), this encounter list has length $m = e \cdot n \cdot (|\mathtt{A}| + |\mathtt{B}|) = 201,600$. The total query latency is $129 + 27 \cdot e \cdot n \cdot (|\mathtt{A}| + |\mathtt{B}|) + (10 + 709) \cdot m = 1.74$ 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 q3, since the naive approach would have to first sort 165.9B records in 2PC and then scan them maintaining the same two Bloom filters.

E 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 MapReduce 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 μ and standard deviation σ, 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, $\mathbb{E}[e^{tX_i}] = t\mu + \frac{1}{2}\sigma^2 t^2$. Hence, $e^{t\mathbb{E}[X]} \leq ne^{(t\mu + \frac{1}{2}\sigma^2 t^2)}$, 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 μ and σ 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 μ_{mmr} and σ_{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 cost_1–cost_4 be the expected costs of the four stages above in the conservative (synchronizing) model. We upper-bound each of these separately.

Estimating 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 μ_1 and σ_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 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 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:

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

Estimating cost$_4$ This cost is immediate:

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

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

F Proofs of Security

In this section, we prove the security of the crypto protocols that comprise CoVault's FE-like construction. The protocol combines many building blocks and therefore we slowly build up to the security proof in §F.6.1, defining several intermediate building blocks and properties.

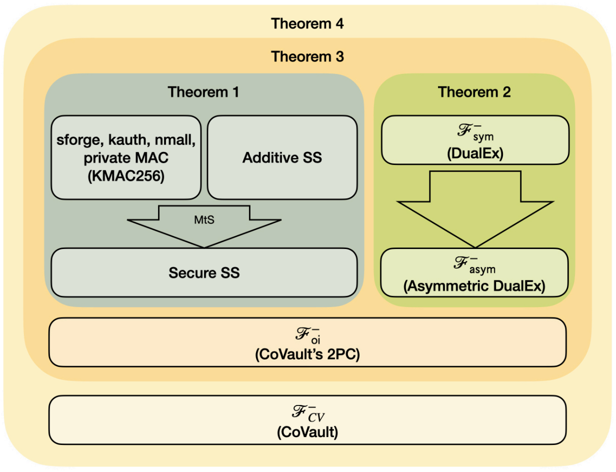
Theorem 4
Theorem 3
Theorem 1
Theorem 2
sforge, kauth, nmall, private MAC (KMAC256)
Additive SS
MtS
Secure SS
$\mathcal{F}^-_{\text{sym}}$ (DualEx)
$\mathcal{F}^-_{\text{asym}}$ (Asymmetric DualEx)
$\mathcal{F}^-_{\text{oi}}$ (CoVault's 2PC)
$\mathcal{F}^-_{CV}$ (CoVault)

Figure 11. An overview of the proof of security of the decryption protocol in CoVault's FE-like construction.

In §F.1, we introduce notation and the preliminaries necessary for our proofs of security: message authentication codes and their properties (§F.2) and secure secret sharing (§F.2.1). In §F.3, we combine MACs and secret sharing to give a construction of a secure secret sharing scheme and a proof of its security. In §F.4, we modify DualEx, a symmetric 2PC 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 §F.5 that a secure secret sharing is secure for computing on authenticated data (Def. 12) even in the presence of one-bit leakage. In §F.6, we define the ideal functionality and provide a proof of security for the decryption protocol. Figure 11 gives and outline of how the building blocks so far fit together.

In §F.7, we show that this functionality can be used to construct a standard functional encryption scheme (with one-bit leakage) where a trusted party, not the encryptor, decides which functions of the plaintext can be decrypted. Finally, in §F.8, we argue that CoVault's *pre-constrained* FE, where encryptors can tie ciphertexts to specific functions at the time of encryption, is secure.

F.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 ℓ of P_i's choice applied to the other party's input x ($|\ell(x)| = 1$). For a randomized functionality, ℓ 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 $n > N$.*

F.2 Message authentication codes

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.

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*

- KeyGen *takes as input a security parameter 1^κ and outputs a random key $k \leftarrow\!\!\$\ \mathcal{D}_\kappa$*
- Mac *takes as input a key k in some domain $\mathcal{D}_\kappa$ associated with a security parameter 1^κ and a message m in some domain $\mathcal{D}_m$ and outputs a tag t.*
- 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 $\top, \perp$.*

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 MAC1-sforge-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 $\mathsf{MAC1\text{-}kauth\text{-}adv}[\mathcal{A}, M] :=$

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

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 MAC1-kauth-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 $\mathsf{MAC1\text{-}nmall\text{-}adv}[\mathcal{A}, M] :=$

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

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 MAC1-nmall-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 MAC-priv-adv$[\mathcal{A}, M] \leq negl(\kappa)$.*

F.2.1 Secure secret-sharing. As mentioned previously, encryption secret-shares data 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 (§F.3).

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

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

F.3 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 we use, 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* Σ*, as follows:*

- Share_M *takes as input a security parameter* 1^κ *and a value x in the domain* $\mathcal{D}_\kappa$ *associated with* κ*. It generates a single key k using* 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* $\mathsf{sh}_i := (x_i, k_i, t)$.
- 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 F.2 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* Σ *an additive secret-sharing scheme, then the MtS secret-sharing scheme* Σ_M *constructed from M and* Σ *is secure.*

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

Authenticity of Σ_M is a bit more unwieldy. We prove the contrapositive that if Σ_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 SS-auth-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 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 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 Share on x, k, t. Now $\mathcal{A}_{\mathsf{nmall}_M}$ runs $\mathcal{A}$ on 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 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 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[64], 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 [64] and Section 6.1 in [45].)

KeyGen**:** Use 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 Σ_M with Σ as the additive secret sharing scheme and M as KMAC256 is secure. In §F.5, we will show that this means it can safely be used inside a secure computation.

F.4 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 we use and its non-standard security guarantees.

We use the DualEx protocol[62] 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 $\mathcal{F}_{\mathrm{sym}}$, 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 decryption functionality, 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 our 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 $\mathcal{F}_{\mathrm{asym}}$ 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 $\mathcal{F}_{\mathrm{asym}}$; when P_1 is malicious, P_1 chooses r.

To do this, we construct a two-party protocol Π that computes $\mathcal{F}_{\mathrm{asym}}$ via access to the symmetric functionality $\mathcal{F}_{\mathrm{sym}}$ 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 $\mathcal{F}_{\mathrm{sym}}$.
- 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. Π *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 Π against malicious adversaries in the $\mathcal{F}^{-}_{\mathrm{sym}}$-hybrid world, we give an adversary $\mathcal{S}_i$ in the ideal world that simulates an execution of Π 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 $\mathcal{F}^{-}_{\mathrm{asym}}$ 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 ℓ 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 ℓ^* 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 $\textsc{ideal}_{\mathcal{S}_1}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the output of the adversary $\mathcal{S}_1$ and the output of the honest party. Concretely,

$$\textsc{ideal}_{\mathcal{S}_1}(1^\kappa, x_1, x_2, \ell, f) = ((y^*, b^*), y_2^*).$$

Hybrid experiment. let $\mathrm{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f)$ 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,

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

$\mathcal{S}_1$ 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 ℓ^*, $b^* = \ell(x_2, y^* \oplus f(x_1', x_2) \oplus r_1'$. 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 ℓ from P_2.
- Forward x_2' and ℓ 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 $\textsc{ideal}_{\mathcal{S}_2}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the output of the honest party and the output of the adversary $\mathcal{S}_2$. Concretely,

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

Hybrid experiment. let $\mathrm{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f)$ 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,

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

$\mathcal{S}_2$ 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, Π 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.

F.5 Secure computation on shared data

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

In our setting, 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 $\mathcal{F}_{\text{oi}}$, 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_Σ 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_Σ (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 sh_i to P_i.

Parties invoke ideal functionality: Party P_i sends an input sh_i' to the ideal functionality $\mathcal{F}$ computing f_Σ; 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 $\textsc{h}^\Sigma_{\mathcal{A}}(1^\kappa, x, f)$ 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 $\mathcal{F}_{\text{oi}}$ that can (only) return $y = f(x)$ (oi stands for "output integrity"). Concretely, the ideal-world execution proceeds as follows:

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

Output delivery: After receiving $f(x)$, the malicious party tells $\mathcal{F}_{\text{oi}}$ 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 $\textsc{ideal}_{\mathcal{S}}(1^\kappa, x, f)$ be the joint random variable containing the output of the adversary and the output of the honest party.

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

However, things are not as simple as that: we use asymmetric DualEx, so we have one bit of leakage to contend with. Specifically, in our case both functionalities $\mathcal{F}$ and $\mathcal{F}_{\text{oi}}$ 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_Σ is defined as in Section F.5), and outputs y to both parties with one-bit leakage.

Define the following protocol Π for computing f_Σ 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 sh_i to P_i.
2. P_i sends an input sh_i' to $\mathcal{F}^-$; the honest party sends the share it received from the dealer. The malicious party additionally sends a leakage function ℓ.
3. $\mathcal{F}^-$ computes $y := f_\Sigma(\mathsf{sh}_1', \mathsf{sh}_2')$ and sends y and ℓ 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* Σ *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 Π 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 Π in the hybrid world.

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

1. $\mathcal{S}$ simulates the dealer by choosing a random value $\hat{x}$ and running $(\hat{\mathsf{sh}}_1, \hat{\mathsf{sh}}_2) \longleftarrow \mathsf{Share}(1^\kappa, \hat{x})$. It gives $\hat{\mathsf{sh}}_i$ to P_i.
2. $\mathcal{S}$ receives $\hat{\mathsf{sh}}_i'$ and a leakage function ℓ from P_i.
3. If $\hat{\mathsf{sh}}_i' \neq \hat{\mathsf{sh}}_i$, then $\mathcal{S}$ sends 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 $\ell^* : \mathcal{X} \longrightarrow \{0, 1\}$, 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 $\textsc{ideal}_{\mathcal{S}}(1^\kappa, x, \ell, f)$ 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,

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

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 $(\mathsf{sh}_1, \mathsf{sh}_2) \longleftarrow \mathsf{Share}(1^\kappa, x)$ and gives sh_i to P_i.

For the adversary $\mathcal{A}$, let $\textsc{h1}_{\mathcal{A}}(1^\kappa, x, f)$ be the joint random variable containing the view of the adversary and the output of the honest party in this hybrid experiment. Concretely,

$$\textsc{h1}_{\mathcal{A}}(1^\kappa, x, \ell, f) = ((\mathsf{sh}_1, \mathsf{sh}_1' \ell, y, b^*, \mathsf{continue/abort}), y_2).$$

Hybrid experiment 2 ($\mathcal{F}^-$-hybrid experiment). Next, we define a second hybrid experiment, substituting ideal functionality $\mathcal{F}^-$ computing f_Σ 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 sh_i' and ℓ from P_i, the experiment fowards this value to $\mathcal{F}^-$ (the honest party sends its true share 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 $\textsc{h2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, f)$ be the joint random variable containing the view of the adversary and the output of the honest party in this hybrid experiment. Concretely,

$$\textsc{h2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, \ell, f) = ((\mathsf{sh}_1, \mathsf{sh}_1' \ell, y', b, \mathsf{continue/abort}), y_2).$$

First, we claim the distribution ensembles $\{\textsc{h1}_{\mathcal{A}}(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\textsc{ideal}_{\mathcal{S}}(1^\kappa, x, f)\}_{\kappa,x,f}$ 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 Σ: $\mathcal{A}'$ 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 Σ is private, so the ensembles must be indistinguishable.

Next, we show the distribution ensembles $\{\textsc{h1}_{\mathcal{A}}(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\textsc{h2}_{\mathcal{A}}^{\mathcal{F}^-}(1^\kappa, x, f)\}_{\kappa,x,f}$ 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 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 Σ. Given sh_i, $\mathcal{A}'$ chooses a random x and receives a share 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 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 Σ.)

This contradicts our assumption that Σ is authenticated, so the ensembles must be indistinguishable.

To conclude, notice that by transitivity, we have computational indistinguishability of $\{\textsc{ideal}_{\mathcal{S}}(1^\kappa, x, f)\}_{\kappa,x,f}$ and $\{\textsc{h2}^{\mathcal{F}^-}_{\mathcal{A}}(1^\kappa, x, f)\}_{\kappa,x,f}$, so Σ is secure for computing on authenticated data in the $\mathcal{F}^-$-hybrid world. □

Notably, if Σ 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.

F.6 Security of the decryption protocol

Finally, we can prove the end-to-end security of the full decryption protocol. Let Σ_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, $\mathsf{Rec}(V_1, V_2) = \{\mathsf{Rec}(\mathsf{sh}^1_k, \mathsf{sh}^2_k)\}_{k=1}^{|V_1|}$ (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 §F.4: $\mathcal{F}^-_{\text{asym}}$ 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 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).

F.6.1 Ideal functionality.

Let Q, QP_1, QP_2 be potentially malicious parties; an adversary can corrupt at most the set $\{Q, QP_i\}$ 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 QP_1, QP_2, respectively. The (trusted) dealer D send sets $V_1, V_2 \in \mathcal{X}^m$, respectively, to QP_1, QP_2. QP_1 and QP_2 send f'_1, f'_2, V'_1, V'_2 to the ideal functionality $\mathcal{F}^-_{CV}$. At this point a corrupted QP_i can additionally send a leakage function ℓ to $\mathcal{F}^-_{CV}$.

Upon receiving $f'_1, f'_2, V'_1, V'_2, \ell$, $\mathcal{F}^-_{CV}$ computes an output y and evaluates ℓ 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 QP. If $b = 1$, it also alerts the honest QP by sending it a distinguished message cheat, in which case that QP aborts.

F.6.2 Full protocol.

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

1. Q sends two (potentially different) functions f_1, f_2 to QP_1, QP_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 an honest QP 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 QP_1, QP_2, respectively, who send inputs V'_1, V'_2 to $\mathcal{F}^-_{\text{asym}}$; an honest QP sends the set it received from the dealer. A malicious QP can additionally send a one-bit leakage function ℓ.
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 QP_i. It also computes ℓ evaluated on the honest party's input and sends the leakage bit to the malicious QP.
4. If the leakage bit is 1, $\mathcal{F}^-_{\text{asym}}$ alerts the honest QP by sending it a distinguished message cheat. Importantly, once cheating is detected, the honest QP completes the current execution but never runs any subsequent iterations of the protocol.
5. Each party QP_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 $y' = \mathsf{Rec}_M(y'_1, y'_2)$.

F.6.3 Proof of security.

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

Proof. Let $\mathcal{A}$ be a PPT adversary corrupting parties Q, QP_i. To prove the security of Π 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, QP_i to simulate an execution of Π 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, ℓ from QP_i. Forward f_{3-i} to QP_{3-i}.
- Send f'_i, V'_i, ℓ to $\mathcal{F}^-_{CV}$; the honest party QP_{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 QP_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 $\textsc{ideal}_{\mathcal{S}}(V, \ell, f)$ be the joint random variable containing the transcript of $\mathcal{S}$ in the ideal world. Concretely,

$$\textsc{ideal}_{\mathcal{S}}(V, \ell, f) = (f_{3-i}, f_i', V_i', \ell, y_i^*, b, y_{3-i}^*).$$

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

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

$\mathcal{S}$ 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 κ bits is at most $2^{-\kappa}$.

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

F.7 Standard functional encryption

Definition 13 (Functional encryption with one-bit leakage). *A functional encryption (FE) scheme with one-bit leakage is a tuple of polynomial-time algorithms $\mathcal{FE}^- = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ such that*

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$ *takes as input a security parameter and outputs a master keypair.*
- $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ *takes as input a master secret key and a function f and outputs a function secret key* sk_f.
- $c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m)$ *takes as input a master public key and a message m and outputs a ciphertext c.*
- $f(m), \ell(m) \leftarrow \mathsf{Dec}(\mathsf{sk}_f, \ell, c)$ *takes as input a function secret key for a function f, a one-bit leakage function $\ell : \mathcal{M} \rightarrow \{0, 1\}$, and a ciphertext of m (for some m), and outputs a value y and $\ell(m)$.*

For correctness, we require that for all messages m, all master keypairs $\mathsf{mpk}, \mathsf{msk} \leftarrow \mathsf{Setup}(1^\kappa)$, *all functions f and corresponding function secret keys* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, *we have* $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Enc}(\mathsf{mpk}, m)) = f(m), \ell(m)$.

We adapt the strong definition[5] of security [54, Def. 3.1] to include one-bit leakage. Following their lead, we call a simulator $\mathcal{S}$ *admissible* if, for each input f, it makes only a single query to $U_m(\cdot)$ and only on the f it received, and *additionally* allow $\mathcal{S}$ to make a single query to $U_m(\cdot)$ for each decryption call on c and only on the leakage function ℓ used in the decryption.

Definition 14 (security (up to one-bit leakage)). *We say an FE scheme with one-bit leakage is* secure *if for all* PPT $\mathcal{A}$ *there exists an admissible stateful* PPT *simulator* $\mathcal{S}$ *such that the following distribution ensembles are computationally indistinguishable:*

$\textit{REAL}_{\mathcal{A}}^{\mathcal{FE}^-}(1^\kappa)$	$\textit{IDEAL}_{\mathcal{A},\mathcal{S}}^{\mathcal{FE}^-}(1^\kappa)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$	$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\kappa)$
$m \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$	$m \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$
$c \leftarrow \mathsf{Enc}(\mathsf{mpk}, m)$	$c \leftarrow \mathcal{S}(\mathsf{mpk}, \mathcal{Q}, 1^{\lvert m \rvert})$
$\alpha \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(c)$	$\alpha \leftarrow \mathcal{A}^{\mathcal{S}^{U_m(\cdot)}(\mathsf{msk},\cdot)}(c)$
output (m, α)	output (m, α)

where $U_m(f) := f(m)$ is a universal oracle for the message m and $\mathcal{Q}$ is the set $\{(f_i, \mathsf{sk}_i, f_i(m))\}_i$ corresponding to $\mathcal{A}$'s queries to the key generation oracle (where sk_i *is function secret key corresponding to f_i).*

A version of FE with one-bit leakage can easily be realized over 2PC and is shown in Construction 1. We note that it meets the definition of FE from the point of view of any one party; of course, if parties collude and share their shares with each other, they can recover m. This case is ruled out by the assumption of our threat model.

[5]See [4] for a comparison of several versions of FE security.

Construction 1 (FE from 2PC). *Let Σ_M be a secure secret sharing scheme, $\Pi_{\mathcal{E}}$ be a CPA-secure encryption scheme, and $\mathcal{F}_{CV}^{-}$ be the ideal functionality defined in F.6.1 using Σ_M.*

- $\mathsf{Setup}(1^\kappa)$*: QP_1, QP_2 each generate a keypair for $\Pi_{\mathcal{E}}$ (a corrupted QP can generate an arbitrary pair (pk_i, sk_i)). Define* $\mathsf{mpk} := (pk_1, pk_2)$ *and* $\mathsf{msk} := (sk_1, sk_2)$.
- $\mathsf{KeyGen}(\mathsf{msk}, f)$*: Send f to each QP_i to receive a function-authorization token pair (f_i, auth_i) such that the authorization token is tied to the function (e.g., a signature on the function description); an honest QP sets $f_i := f$, and a corrupted QP outputs an arbitrary pair of bit strings. Output $sk_f := ((f_1, \mathsf{auth}_1), (f_2, \mathsf{auth}_2))$.*
- $\mathsf{Enc}(\mathsf{mpk}, m)$*: parse* $\mathsf{mpk} = (pk_1, pk_2)$*, compute* $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}_M(m)$*, and set $c_i \leftarrow \Pi_{\mathcal{E}}.\mathsf{Enc}(pk_i, \mathsf{sh}_i)$. Output $c := (c_1, c_2)$.*
- $\mathsf{Dec}(\mathsf{sk}_f, \ell, c)$*: parse* $\mathsf{sk}_f = ((f_1, \mathsf{auth}_1), (f_2, \mathsf{auth}_2))$ *and $c = (c_1, c_2)$. Send $(f_i, \mathsf{auth}_i), c_i$ to QP_i. An honest QP validates its authorization token* auth_i *and, if the check passes, computes* $\mathsf{sh}_i \leftarrow \Pi_{\mathcal{E}}.\mathsf{Dec}(sk_i, c_i)$ *and feeds f_i, sh_i into $\mathcal{F}_{CV}^{-}$. A corrupted QP picks its inputs to $\mathcal{F}_{CV}^{-}$ arbitrarily and also sends a leakage function ℓ'. $\mathcal{F}_{CV}^{-}$ outputs y to both QPs and additionally sends ℓ' evaluated on the honest QP's input to the corrupted QP. The algorithm outputs y and the message leakage $\ell(m)$ which is implicitly defined as $\ell(m) := \ell'(m \oplus \mathsf{sh}_i)$*[6]*, where* $\mathsf{sh}_i \leftarrow \Pi_{\mathcal{E}}.\mathsf{Dec}(sk_i, c_i)$ *and QP_i is the corrupted QP.*

Correctness follows from the correctness of Σ_M, $\Pi_{\mathcal{E}}$, and $\mathcal{F}_{CV}^{-}$: if all parties behave honestly, $\mathcal{F}_{CV}^{-}$ will output $y = f(\mathsf{Rec}_M(\Pi_{\mathcal{E}}.\mathsf{Dec}(pk_1, c_1), \Pi_{\mathcal{E}}.\mathsf{Dec}(pk_2, c_2)) = f(\mathsf{Rec}_M(\mathsf{sh}_1, \mathsf{sh}_2)) = f(m)$.

Theorem 5. *Construction 1 is secure up to one-bit leakage in the $\mathcal{F}_{CV}^{-}$-hybrid model.*

Proof. Let $\mathcal{A}$ be a PPT adversary corrupting QP_i. We give a simulator $\mathcal{S}_i$ such that the distribution ensembles $\{\text{REAL}_{\mathcal{A}}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ and $\{\text{IDEAL}_{\mathcal{A},\mathcal{S}_i}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ are computationally indistinguishable (in fact, they are perfectly indistinguishable). For simplicity of notation, we assume the base secret-sharing scheme Σ of Σ_M is the XOR scheme; our proof generalizes to any other underlying scheme as long as it is possible, given ℓ' and sh_i, to compute a function ℓ such that $\ell(x) := \ell'(y)$ for all (x, y) such that $\Sigma.\mathsf{Rec}(y, \mathsf{sh}_i) = x$. We abuse notation slightly by using sh_i to refer both to fully authenticated shares of Σ_M and the underlying shares of Σ without authenticating information.

Simulator $\mathcal{S}_i$:

1. Given mpk, Q, and $1^{|m|}$, $\mathcal{S}_i$ samples a uniform m^*, and runs Enc honestly: it computes $(\mathsf{sh}_1^*, \mathsf{sh}_2^*) \leftarrow \mathsf{Share}_M(m^*)$ and outputs $c := (\Pi_{\mathcal{E}}.\mathsf{Enc}(pk_1, \mathsf{sh}_1^*), \Pi_{\mathcal{E}}.\mathsf{Enc}(pk_2, \mathsf{sh}_2^*))$.
2. After this point, it responds to any key-generation queries f honestly, namely with $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$. Whenever $\mathcal{A}$ runs Dec on c, $\mathcal{S}_i$ simulates the underlying ideal functionality $\mathcal{F}_{CV}^{-}$, intercepting the inputs sh_i', ℓ'. If $\mathsf{sh}_i' \neq \mathsf{sh}_i^*$, $\mathcal{S}_i$ sets $y^* :=\perp$; otherwise $y^* := U_m(f_i)$. It sets $b^* := U_m(\ell)$, where $\ell(m) := \ell'(m \oplus \mathsf{sh}_i^*)$ is the implicit leakage function of Dec. $\mathcal{S}_i$ returns y^*, b^* to $\mathcal{A}$.

Clearly, $\mathcal{S}_i$ is admissible. We now consider a series of hybrid-world executions to show that the distribution ensembles $\{\text{REAL}_{\mathcal{A}}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ and $\{\text{IDEAL}_{\mathcal{A},\mathcal{S}_i}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ are indistinguishable.

Hybrid 1 Our first hybrid behaves in the same way as the simulator $\mathcal{S}_i$ except that it also computes a true sharing $(\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}_M(m)$ and computes c as $(\Pi_{\mathcal{E}}.\mathsf{Enc}(pk_i, \mathsf{sh}_i^*), \Pi_{\mathcal{E}}.\mathsf{Enc}(pk_{3-i}, \mathsf{sh}_{3-i})$.

Hybrid 2 In the next hybrid, we also change the remaining ciphertext component so $c := (\Pi_{\mathcal{E}}.\mathsf{Enc}(pk_i, \mathsf{sh}_i), \Pi_{\mathcal{E}}.\mathsf{Enc}(pk_{3-i}, \mathsf{sh}_{3-i})$.

Hybrid 3 Finally, instead of simulating $\mathcal{F}_{CV}^{-}$ during Dec queries, our last hybrid simply participates honestly in the decryption (and thus the evaluation of $\mathcal{F}_{CV}^{-}$).

The indistinguishability of $\{\text{IDEAL}_{\mathcal{A},\mathcal{S}_i}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ and $\{\text{H1}_{\mathcal{A}}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ follows immediately from CPA-security of $\Pi_{\mathcal{E}}$, since $\mathcal{A}$ does not know sk_{3-i}. Hybrid 1 and 2 are indistinguishable by reduction to the privacy of the secret-sharing scheme Σ_M. In more detail, a distinguisher $\mathcal{D}$ between the hybrids could be used to construct an adversary $\mathcal{A}'$ with nonnegligible advantage in the privacy game of Σ_M: $\mathcal{A}'$ simply lets $x_0 := m$ and samples x_1 uniformly at random. Upon receiving a share from the game, it runs $\mathcal{D}$ with c_i set to the encryption under pk_i of the share it received from the game (sh_i or sh_i^*); if $\mathcal{D}$ outputs "real", it outputs 0, and otherwise it outputs 1.

Next, notice that Hybrid 2's output y^* equals $f_\Sigma(\mathsf{sh}_i, m \oplus \mathsf{sh}_i)$, which is also the output of $\mathcal{F}_{CV}^{-}$ in Hybrid 3. Additionally, by definition we have $b^* = \ell'(m \oplus \mathsf{sh}_i^*)$, while in Hybrid 3, $\mathcal{F}_{CV}^{-}$ outputs $b = \ell'(\mathsf{sh}_{3-i}) = \ell'(m \oplus \mathsf{sh}_i)$. We just argued above that

[6]This conversion between ℓ and ℓ' is necessary because we defined $\mathcal{F}_{CV}^{-}$ as evaluating the leakage function on *the other party's input*, while here we define it to be evaluated on the *message*.

by the privacy of Σ_M, the distributions of sh_i^* and sh_i are indistinguishable, and therefore so are b^* and b. Finally, note that Hybrid 3 is exactly the real-world experiment.

In summary, by the privacy of Σ_M, CPA-security of $\Pi_{\mathcal{E}}$, and the security of $\mathcal{F}_{CV}^{-}$, the distribution ensembles $\{\text{REAL}_{\mathcal{A}}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ and $\{\text{IDEAL}_{\mathcal{A},\mathcal{S}}^{\mathcal{FE}^-}(1^\kappa)\}_\kappa$ are (perfectly) indistinguishable. □

F.8 CoVault's functional encryption

In our setting, we want a slightly stronger version of functional encryption: the data sources should be able to fix the functions allowed to run on its data, and by what queriers those functions can be run. We can capture this by modifying the above definition of functional encryption as follows.

We call this *pre-constrained functional encryption* (PCFE), to mirror the recently introduced notion of pre-constrained encryption [7].[7] Because we believe PCFE to be an independently interesting primitive, below we give a definition without one-bit leakage; it is easy to adapt it to include leakage as in Definition 13 by letting Dec additionally accept an input ℓ and output $\ell(m)$.

Definition 15 (Pre-constrained functional encryption). *A pre-constrained FE scheme is a tuple of polynomial-time algorithms* $\mathcal{PCFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *such that*

- $(\mathsf{mpk}[F,P], \mathsf{msk}[F,P]) \leftarrow \mathsf{Setup}(1^\kappa, F, P)$*: the setup algorithm takes as input a security parameter, a function class F, and a set of public keys P representing authorised parties; it outputs a constrained master keypair.*
- $\mathsf{sk}_{f,pk} \leftarrow \mathsf{KeyGen}(\mathsf{msk}[F,P], f, pk^8)$*: key generation takes as input a constrained master secret key, a function f, and a party's public key pk. If $f \in F$ and $pk \in P$, it outputs a party-specific function secret key $\mathsf{sk}_{f,pk}$; otherwise it outputs $\perp$.*
- $c \leftarrow \mathsf{Enc}(\mathsf{mpk}[F,P], m)$*: encryption takes as input a constrained master public key and a message m and outputs a ciphertext c.*
- $f(m) \leftarrow \mathsf{Dec}(\mathsf{sk}_{f,pk}, c)$*: decryption takes as input a function secret key for a function f and a ciphertext and outputs $f(m)$ for some message m (if $\mathsf{sk}_f = \perp$ it outputs $\perp$).*

For correctness, we require that for all messages m, all function classes F and sets of parties P, all master keypairs $\mathsf{mpk}[F,P], \mathsf{msk}[F,P] \leftarrow \mathsf{Setup}(1^\kappa, F, P)$*, all functions $f \in F$ and parties $pk \in P$, and all function keys* $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}[F,P], f, pk)$*, we have* $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Enc}(\mathsf{mpk}[F,P], m)) = f(m)$.

The security definition is a straightforward extension of Definition 14, and we define the class of admissible simulators in the same way. This time, however, our construction will use trusted execution environments (TEEs). We assume the TEEs work perfectly, and leave the question of modelling our TEE architecture as an interesting open problem. Interested readers may look at [50] for an example of how the TEEs could be modelled.

Definition 16 (security (with one-bit leakage)). *We say a pre-constrained FE scheme with one-bit leakage is* secure *if for all* PPT $\mathcal{A}$ *there exists an admissible stateful* PPT *simulator* $\mathcal{S}$ *such that the following distribution ensembles are computationally indistinguishable:*

```
$\underline{\text{REAL}_{\mathcal{A}}^{\mathcal{PCFE}-}(1^\kappa, F, P)}$
$(\mathsf{mpk}[F,P], \mathsf{msk}[F,P]) \leftarrow \mathsf{Setup}(1^\kappa, F, P)$
$m \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}[F,P],\cdot)}(\mathsf{mpk}[F,P])$
$c \leftarrow \mathsf{Enc}(\mathsf{mpk}[F,P], m)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}[F,P],\cdot)}(c)$
output $(m, \alpha)$
```

```
$\underline{\text{IDEAL}_{\mathcal{A},\mathcal{S}}(1^\kappa, F, P)}$
$(\mathsf{mpk}[F,P], \mathsf{msk}[F,P]) \leftarrow \mathsf{Setup}(1^\kappa, F, P)$
$m \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}[F,P],\cdot)}(\mathsf{mpk}[F,P])$
$c \leftarrow \mathcal{S}(\mathsf{mpk}[F,P], Q, 1^{|m|})$
$\alpha \leftarrow \mathcal{A}^{\mathcal{S}^{U_m(\cdot)}(\mathsf{msk}[F,P],\cdot)}(c)$
output $(m, \alpha)$
```

The following is CoVault's FE construction.

[7]Unlike [7], we do not give a full treatment of this primitive but only a definition which is a straightforward extension of existing definitions and a simple construction from strong assumptions. We leave a more thorough exploration of pre-constrained FE and constructions from standard assumptions as an interesting direction for future work.

[8]This is intentionally not styled pk to show that pk and sk_f are unrelated; pk is a public key for a completely different (underlying) PKE.

Construction 2 (PCFE from 2PC and TEEs)**.** *Let $\mathcal{F}_{CV}^{-}$ and Σ_M be defined as in Construction 1.* $\mathsf{Enc}(\mathsf{mpk}[F, P], m)$ *is defined the same way as before, and the remaining algorithms work as follows:*

- $\mathsf{Setup}(1^\kappa, F, P)$ *produces and signs a TEE configuration* config *identifying F and P and provisions (a set of) TEEs with it; as before, it outputs a garbage* mpk*,*
- $\mathsf{KeyGen}(\mathsf{msk}[F, P] := \mathsf{config}, f, pk)$ *checks if $f \in F$ and $pk \in P$, where F, P are defined by* config*; if so, it outputs a function-authorization token pair* (f, auth)*, and*
- $\mathsf{Dec}(sk_f := (f, \mathsf{auth}), (c_1, c_2))$ *validates the authorization token* auth*. If the check passes, it runs $\mathcal{F}_{CV}^{-}$ like before; otherwise, it outputs $\perp$.*

In practice, Setup is run by every provisioning service (PS) for the TEEs that act as data processors (DPs) in its own pipeline. (The PSs themselves are attested by community volunteers.) Enc is run locally by the data sources, and the resulting ciphertexts (encrypted shares) are each sent to a different DP via secure point-to-point channels (i.e., using a PKE). KeyGen and Dec are combined and run inside an MPC protocol by the set of DPs provisioned with the configuration produced by the setup, using $\mathcal{F}_{CV}^{-}$ as before and with the input pk set to the querier's public key.

Correctness holds by the same logic as before. Since we assume the TEEs behave as intended, and the checks on f and pk are always computed correctly; therefore, on the set of inputs $f \in F$ and $pk \in P$, this construction is equivalent to Construction 1, and security (with one-bit leakage) follows as before.