HypoCert: Certified Hypothesis Testing

Sacha Servan-Schreiber
Brown University
Rhode Island, United States
aservans@brown.edu

Olga Ohrimenko
Microsoft Research
Cambridge, United Kingdom
oohrim@microsoft.com

Tim Kraska
MIT CSAIL
Massachusetts, United States
kraska@mit.edu

Emanuel Zgraggen
MIT CSAIL
Massachusetts, United States
emzg@mit.edu

ABSTRACT

HypoCert is a novel approach to the complex issue of preventing "p-hacking" in scientific studies and provides a method for controlling false-discoveries in scientific fields. We eliminate any potential for bias on the part of researchers during data-analysis and provide guarantees on the validity of each statistical test performed on a dataset. HypoCert uses cryptographic techniques to certify outcomes of statistical tests by a decentralized authority and a blockchain to audit this process. We present three separate theoretical constructions for realizing HypoCert, each with their own benefits, and fully implement and evaluate a construction based on multi-party computation with secret sharing.

PVLDB Reference Format:
Sacha Servan-Schreiber, Olga Ohrimenko, Tim Kraska, Emanuel Zgraggen. HypoCert: Certified Hypothesis Testing. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/TBD

1. INTRODUCTION

"Data is the new oil" and as such, it is mined (i.e., gathered), shared, analyzed, re-analyzed, and re-re-analyzed until the data at hand yields to more and more interesting insights. With every exploration to find yet another insight, the chance of encountering a random correlation increases. This phenomena is formally known as the multiple comparisons problem (MCP) and, if done in a systematic fashion, is often referred to as "HARKing" [37], "p-hacking" [28] and "data dredging".

While a variety of statistical techniques exist to control for the MCP [17, 3], there is surprisingly almost no support to ensure that analysts actually use them. Rather individual research groups rely on often varying data analysis guidelines and trust in their group members to follow them. Things get even worse when the same data is analyzed by several institutions or teams. It is currently close to impossible to reliably employ statistical procedures that avoid any form of "p-hacking" across collaborators. It only requires one member to "misuse" the data (intentional or not) and detecting, let alone recovering from such incidents is next to impossible. This problem is perhaps amplified by three factors: 1) the pressure on PhD students and PIs to publish, 2) "publication bias" [16] as papers with significant results are more likely to be published, and 3) the increasing trend to share and make datasets publicly available for any researchers to use. It is therefore unsurprising that MCP is among the leading reasons why the scientific community is plagued by false discoveries[2, 30, 33, 31].

To illustrate this problem further consider a publicly available dataset such as MIMIC III [35]. This dataset contains de-identified health data associated with $\approx 40,000$ critical care patients. MIMIC III has been already used in various studies [40, 25, 29] and it is probably one of the most (over)analyzed clinical datasets. As such, any discovery made on MIMIC runs the risk of being a false discovery. Even if a particular group of researchers follow a proper MCP protocol, there is no control happening across different groups and tracking hypotheses at a global scale poses many challenges. It is therefore hard to judge the validity of any insight derived from such a dataset.

A solution to guarantee validity of insights commonly used in clinical trials - preregistration of hypotheses [12] - falls short in these scenarios. The data is collected upfront without knowing what kind of analysis will be done later on. Perhaps more promising is the use of a hold-out dataset. The MIMIC authors could have released only 30K patient records as an exploration dataset (EDS) and hold back 10K as a validation dataset (VDS). The EDS can then be used in arbitrary ways to find interesting hypothesis. However, before any publication is made by a research group using the dataset, the hypothesis can be tested for its statistical significance over the VDS. Unfortunately, in order to use the VDS more than once, the same requirements as before holds true: every hypothesis over the VDS has to be tracked and controlled for. Furthermore, the data owner, the MIMIC authors in this case, need to provide this hypothesis validation service. This is both a burden for the data owners as well as a potential source of bias. Researchers need to trust the data owners to apply MCP control procedures correctly and to objectively evaluate their hypothesis.

This example illustrates the motivation behind HypoCert. The goal of our work is to create a system that guarantees the validity of outcomes of statistical tests and allows readers

Proceedings of the VLDB Endowment, Vol. 12, No. xxx

DOI: https://doi.org/TBD

or reviewers of publications to audit them. Using proven cryptographic techniques to certify outcomes of statistical tests by a decentralized authority, we eliminate any potential for bias on the part of researchers or data owners. HypoCert can be used in various settings, including cases where the data is public and only the hold-out data is fed into HypoCert (as in the example above), in smaller settings were a few research groups collaborate on combined data or even within single teams where lab managers can opt to encrypt all of their data and use HypoCert as a way to prevent unintentional bias, assign accountability and foster reproducibility.

HypoCert. Figure 1 shows a high-level overview of our system. (1) A data owner encrypts their dataset (either the full dataset or just a hold-out) and submits it to HypoCert: a decentralized platform consisting of multiple nodes that are run by different entities (e.g., universities or research groups). (2a and 3a) Researchers can post request for statistical tests to HypoCert. (2b and 3b) HypoCert securely computes these tests, either using homomorphic encryption or secret sharing cryptographic techniques, and stores the results (and transcript of the computation) in order on a tamperproof log (e.g., a blockchain). (4) One of the researchers decides to publish their finding and includes a "certified p-value" in their paper. (5) A reader or auditor of this publication can query the blockchain, retrieve all p-values of all the tests that have been run on this particular dataset and apply an incremental MCP control procedure [50, 20] to validated the publication's finding. HypoCert prevents p-hacking by using encrypted data and by guaranteeing that every statistical test is accounted for.

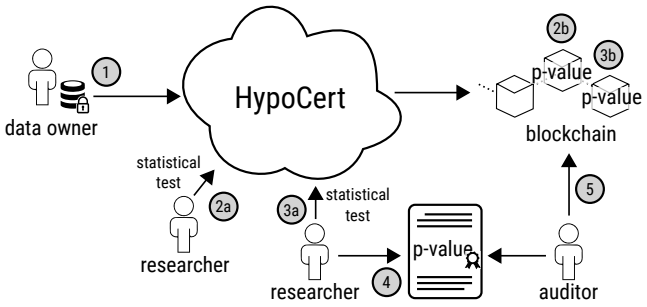


Figure 1: High-level overview of HypoCert.

HypoCert is, to the best of our knowledge, the first cryptographically based solution to the problem of "p-hacking" with provable security guarantees. In this paper we discuss three separate theoretical constructions for realizing HypoCert, that differ in the underlying cryptographic techniques they rely on, and highlight benefits and drawbacks of each. We implement one of these constructions, which uses multi-party computation and supports three widely used hypothesis tests (Student's T-test, Pearson Correlation, and Chi-Squared), and evaluate its performance with various configurations and dataset sizes.

2. PROBLEM STATEMENT

We consider a setting where a data owner wants to make her dataset $\mathcal{D}$, where $\mathcal{D}$ is either a full dataset or just a hold-out, available to a set of researchers, denoted as parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, for executing statistical tests in a truthful manner, that is, reporting test results that are correctly adjusted based on all previously run tests. Additionally, only the party interested in performing a particular test should participate in its computation while requiring minimal interaction with the other parties, including the data owner. The last constraint is inspired by how these tests are computed today: the data owner releases her dataset to researchers and goes offline while each researcher can individually run their tests against their copy of the data.

HypoCert addresses the above requirements. It does so by restricting the exposure of the dataset $\mathcal{D}$ to the parties. That is, the parties can run tests on $\mathcal{D}$ only through HypoCert's interface which as we will see later is implemented in a decentralized manner. Crucially, for every test that HypoCert executes, it produces a publicly available certificate ψ that records the information about the test. A certificate of a test is a tuple $(\tau, \mathcal{P}, C, \mathtt{p})$ where τ is the *test index* (i.e., its order among all the tests that have been executed by HypoCert on $\mathcal{D}$), $\mathcal{P}$ is the identifier of the researcher that requested the test (e.g., his public key), C is the code of the statistical test, and $\mathtt{p}$ is a result of executing $C(\mathcal{D})$. In order to control for multiple hypotheses (and prevent errors due to the MCP) one can adjust the result $\mathtt{p}$ of τth test by using a list of certificates with test indices $1, 2, \ldots, \tau - 1$ and applying incremental variants of procedures that bound the marginal false discovery rate at a specified α level [50, 20]. Note that standard MCP procedures such as Bonferroni [17] are not applicable here as they require knowledge about all the hypotheses being evaluated upfront, whereas in our context, the hypotheses are generated incrementally.

HypoCert aims to provide the following properties:

- **Confidentiality:** Information about $\mathcal{D}$ is released only through results of the code executed by HypoCert.
- **Access control:** The tests on $\mathcal{D}$ are executed only by the HypoCert system and only a pre-approved set of researchers can request HypoCert to execute these tests.
- **Transparency:** Every test executed by HypoCert has a certificate that is publicly accessible.
- **Audit:** Given a certificate $\psi = (\tau, \mathcal{P}, C, \mathtt{p})$, anyone can verify that (1) $\mathtt{p}$ is the output of $C(\mathcal{D})$, (2) C is a test requested by researcher $\mathcal{P}$, and (3) exactly τ tests have been executed since the data owner released $\mathcal{D}$.

We note that the certificate by itself does not ensure that $\mathtt{p} = C(\mathcal{D})$, however, HypoCert records sufficient information that anyone can verify whether it is the case, thereby, exposing malicious behavior.

Trust model. HypoCert provides the above properties in the following trust model.

We assume that the data owner is trusted and does not collude with the researchers who run the statistical tests. This is a necessary assumption given that a malicious data owner has unfettered access to the dataset.

We assume that all parties are *honest-but-curious*, that is they individually run their part of the protocol correctly but are interested in learning more about the underlying dataset (i.e., learning more than what is available from the output produced by HypoCert). Moreover, they may try to request HypoCert to execute mal-formed code that is not a statistical

test (e.g., ask for a partial content of $\mathcal{D}$). We assume that at most $\lfloor \frac{n-1}{2} \rfloor$ parties may be colluding with each other in order to obtain more information about the dataset or change the test results but that the set of colluding parties is static (does not change throughout protocol execution).

We assume a public key infrastructure in place which allows to identify individual parties (researchers) by their public key. To this end, the messages exchanged in the protocol are digitally signed and can be verified against party's identity.

Our protocol makes use of a blockchain abstraction that we assume is always available. We provide a more detailed description of the necessary requirements of this abstraction in the following section.

3. SYSTEM OVERVIEW

Our design of HypoCert is based on the following two observations. First, the data owner cannot release a dataset $\mathcal{D}$ to the researchers "in the clear" since it creates a possibility for both intentional and unintentional bias and "p-hacking" (e.g., creates a possibility for parties to run tests privately and report only favorable results). Hence, the data has to be stored and computed on in such a way that only the output of the statistical test performed on $\mathcal{D}$ is made available to the parties. Second, an oracle that simply returns the results is also not sufficient for enforcing correctness as it can be misused by parties querying it until a favorable result is obtained while avoiding to report the intermediate queries. To address these observations, HypoCert makes use of *secure computation*, that executes tests on an encoded form of $\mathcal{D}$ without revealing any information about $\mathcal{D}$ except the test output, and a *blockchain*, Log, that records the sequence of statistical tests performed on $\mathcal{D}$ by the parties. Together these two functionalities guarantee that all tests executed against $\mathcal{D}$ are recorded.

Secure Computation. HypoCert is trivial to implement if there exists a trusted third party that the owner is willing to release the dataset to. Alas, this is often not the case. To this end, we explore three ways of instantiating secure computation with cryptographic techniques which distributes the trust among the parties. We describe these secure computation methods in the next section and here identify the main functionality required by HypoCert. Secure computation provides a way for the data owner (1) to share her data with the parties in an encoded form (either encrypted or secret shared) such that a majority of the parties are required to recover the encoded data and (2) allows parties to compute on the encoded data without learning information about $\mathcal{D}$.

Blockchain. HypoCert relies on a blockchain that we model as tamperproof log abstraction, Log. It records tests run against the dataset, their results, order of execution, and who requested them. The blockchain can be maintained by a central authority (e.g., the data owner), by the parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ themselves using a consensus protocol (e.g., Paxos [38]), or by independent parties (e.g., using a public blockchain based on a decentralized consensus protocol such as Nakamoto consensus [41]). Similar to the ledger abstraction used in [10], our main requirement is that the above abstraction is correct and always available.

For our purposes, Log provides a very simple functionality that stores a list of messages (tuples) that can be appended to Log but cannot be modified in any other way. Parties can append to the log and read from it. Every message that is persisted on the log is assigned a unique sequential identifier. Depending on the scenario, the log can be also read by the data owner, a third party such as an auditor, or be available publicly. We ensure that the parties post to Log messages signed with their public key that allows everyone with access to the log to verify them. We assume that messages posted to the log are broadcast to the parties (e.g., if the log is not maintained by the parties themselves an event trigger could be set).

Outline. We are now ready to present an outline of how HypoCert operates.

The data owner, $\mathcal{O}$, interacts with the system only during the setup. She shares with the parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ the (encoded) dataset $\mathcal{D}$ and identities of the researchers that can query $\mathcal{D}$ (e.g., their public keys). The data owner also distributes enough information to the parties such that a majority of them in collaboration can decode the data (e.g., when encryption is used the data owner distributed shares of a secret key used to encode $\mathcal{D}$). The data owner then initializes Log. If a public blockchain is used then no initialization is required and the data owner simply posts to Log a signed message $(0, \mathcal{O}, \perp, \perp)$ that corresponds to the counter of the tests to be executed on $\mathcal{D}$ set to zero. Otherwise, the parties initialize a distributed empty log with $(0, \mathcal{O}, \perp, \perp)$ being the genesis message. We note that the log is used to store test certificates as well as auxiliary information (e.g., execution transcript) that facilitates secure computation and auditing. The data owner then goes offline and the system is ready to execute statistical tests.

During test computation, a researcher, say $\mathcal{P}$, specifies the test C that it wants to execute on $\mathcal{D}$. We note that $\mathcal{P}$ may be one of the parties in $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ or anyone approved by the data owner during the setup of HypoCert. $\mathcal{P}$ posts a message $(\mathcal{P}, C)$ to Log and obtains the *test index*, τ. If C is large, $\mathcal{P}$ uploads only a hash of C and stores C in a public repository (recall that it is not our goal to hide test code since it is required for audit and test certificates). For simplicity, we assume that only one test is executed at a time. It is not a necessary requirement since the dataset is static and Log ensures that every test is assigned a sequential test index.

$\mathcal{P}$ then executes the test on the encoded dataset; depending on the secure computation approach this may require interaction between the parties. Once the test is computed, the result of the test, p, is available only in an encoded form. Recall that the test result cannot be recovered by *any* of the parties individually and requires a majority of the parties to reveal it. To this end, $\mathcal{P}$ requests help from $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ by posting a message to Log. Each party $\mathcal{P}_j$ verifies that the message indeed came from one of the researchers allowed to run the tests and posts to Log his partially recovered p. This specification ensures that each test result is made publicly available (to all parties). Once sufficient number of parties have posted their messages and $\mathcal{P}$ recovers p, he adds a message $(\tau, \mathcal{P}, C, \mathsf{p})$ to Log and uses it as the certificate of the test. Since several secure computation techniques require interaction between the parties (e.g., for recovering the result) every party is required to include test index τ into messages that are related to τth test.

During an audit of a test with certificate $(\tau, \mathcal{P}, C, \mathsf{p})$, one retrieves all the messages related to τ from Log. It verifies

the signatures on all the messages, including the certificate, and re-executes C. We require that Log contains sufficient information to verify whether $\mathtt{p} = C(\mathcal{D})$ but not learn any additional information about $\mathcal{D}$.

We note that during test computation, $\mathcal{P}$ may request the parties to decrypt $\mathtt{p}'$ that does not correspond to a correct execution of C. Though, this behaviour will be discovered later during an audit, HypoCert can prevent this from happening by requesting every party to execute C locally before starting to decrypt. (Indeed, our instantiation of HypoCert based on secret sharing in Section 4.3 does so implicitly).

4. HypoCert INSTANTIATIONS

We explore three high-level instantiations that differ in the underlying secure computation method. Since these methods also require different level of interaction between the parties, they also differ in the number of messages posted by the parties to the log. The first two methods are based on threshold homomorphic encryption while the third uses secret sharing. We compare pros and cons of each method in Table 1.

4.1 Threshold FHE-based Instantiation

We begin by describing the "ideal" instantiation of HypoCert that relies on threshold fully-homomorphic encryption. See Figure 2 for an illustration of this scheme.

Threshold Fully Homomorphic Encryption. Fully homomorphic encryption (FHE) allows one to evaluate arbitrary functions (represented as circuits) on encrypted data without decrypting the data [44, 24]. The first FHE construction is due to Gentry [23] and is based on lattices. Many FHE works are built in the setting where the party that generated the public/secret key pair of the FHE scheme is the one that decrypts the result of the FHE computation [39, 46]. However, HypoCert's setting is different in that the data owner goes offline once the keys are generated and the dataset is encrypted. To this end, we rely on threshold fully homomorphic encryption (TFHE) that distributes the decryption functionality amongst a set of parties.

Threshold fully homomorphic encryption [32, 1] generates a single public key and n secret key shares that are distributed to the parties accordingly. Anyone who has access to the public key can encrypt and compute over the encrypted data. However, only a set of parties with access to the secret key shares can decrypt the data, including a result of a computation. For our purposes, we assume that a majority of the parties is required in order to decrypt the result though more complex access structures can be supported [32]. TFHE is semantically secure and nothing can be learned from ciphertexts (including from computations thereof) with $\lfloor \frac{n-1}{2} \rfloor$ or less secret key shares. We use the definition of a TFHE scheme from [32] that consists of five algorithms: (Setup, Encrypt, Eval, PartDec, FinDec) and adopt it to HypoCert's setting.

TFHE HypoCert. During the setup phase, the trusted data owner executes algorithm Setup that outputs a public key pk and secret key shares $\mathsf{sk}_1, ..., \mathsf{sk}_n$. The data owner then runs $\mathsf{Encrypt}(\mathsf{pk}, \mathcal{D})$ on dataset $\mathcal{D}$ that returns encoded dataset $[\mathcal{D}]$, where every value of $\mathcal{D}$ is individually encrypted. The data owner sends $\mathsf{pk}, \mathsf{sk}_i, [\mathcal{D}]$ to $\mathcal{P}$. The owner also announces the identity of every party i that can run the protocol (e.g., it can be the verification key of a digital signature that will be used to verify i's signatures). If an audit by a third party is required, $[\mathcal{D}]$ is shared with an auditor.

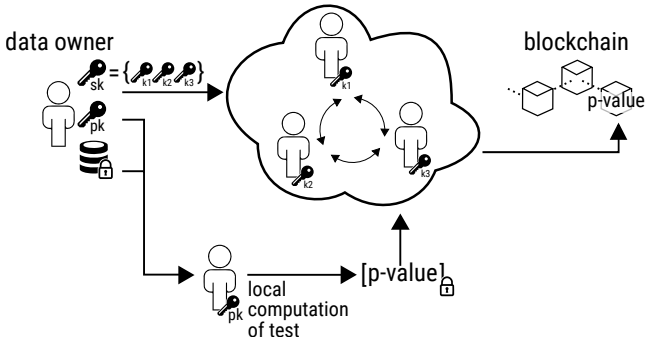


Figure 2: Overview of the TFHE based HypoCert instantiation (Section 4.1). The researcher receives an encrypted dataset from the data owner and proceeds to compute statistical tests locally, using parties in the HypoCert network to decrypt the obtained p-value and certify it on a blockchain Log.

Let $\mathcal{P}$ be the researcher that wishes to run a statistical test C (represented as a Boolean or arithmetic circuit). $\mathcal{P}$ posts $(\mathcal{P}, C)$ to Log and obtains τ. (This step is the same for the other two instantiations, hence, we omit it in the next two sections). He then runs the deterministic algorithm $\mathsf{Eval}(C, [\mathcal{D}])$ to evaluate the circuit using $\mathcal{D}$ as input. The output of this algorithm is the encrypted result of the statistical test $[\mathtt{p}]$. $\mathcal{P}$ adds a message $(\tau, i, C, [\mathtt{p}])$ to Log and a message containing its partial decryption of $\mathtt{p}$.

Every party $\mathcal{P}_j$, upon receiving $(\tau, i, C, [\mathtt{p}])$ verifies that it was signed by a valid party. If the verification succeeds, it executes $\mathsf{PartDec}([\mathtt{p}], \mathsf{sk}_j)$ that outputs $\mathcal{P}_j$'s partial decryption of $[\mathtt{p}]$, $\mathtt{p}_j$. Party j then posts $(\tau, \mathtt{p}_j)$ to Log. $\mathcal{P}$ waits until at least $\lfloor n/2 \rfloor + 1$ parties have posted their partial decryptions to the log. He then runs $\mathsf{FinDec}(B)$ to decrypt $[\mathtt{p}]$ where B is a sequence of all partial decryptions that correspond to τth test. He then posts $(\tau, \mathcal{P}, C, \mathtt{p})$ to Log.

Optimizations. We note that $\mathcal{P}$ can use a cloud provider to store the encrypted dataset and run Eval. In this case, during the setup the data owner sends to each party only the keys and posts the hash of the encrypted dataset to the Log, allowing the execution of Eval to be audited.

Security. We now argue that TFHE HypoCert satisfies properties outlined in Section 2.

- **Confidentiality:** $\mathcal{D}$ is available only in an encrypted form and can be decrypted only by the majority of the parties. The parties perform decryption only when one of the parties requests decryption via a message recorded on Log.
- **Access control:** Though anyone can homomorphically compute on $[\mathcal{D}]$, the obtained encrypted result cannot be decrypted unless a majority of $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ colludes. Moreover, each party performs a partial decryption of a result only if the decryption request message was posted on Log and signed by one of the ap-

proved parties. Hence, the decryption happens only if a party from $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ requested a decyrption.

- **Transparency:** This property is guaranteed again by the fact that a majority of the parties are required to decrypt a result. That is, even if someone computes on $[D]$, the obtained result is encrypted and will not be decrypted unless a majority of the parties colludes. However, every test whose result p is decrypted has partial decryption shares of p stored on the log. Hence, even if $\mathcal{P}$ that initiated the test goes offline after requesting a decryption of p, the certificate can be reconstructed from Log using FinDec since it can be run by anyone with access to Log
- **Audit:** A correct certificate is a signed message $(\tau, \mathcal{P}, C, \mathsf{p})$ that is stored on Log. Since Eval is a deterministic, anyone with access to $[\mathcal{D}]$ can obtain $[\mathsf{p}']$ by running $\mathsf{Eval}(C, [\mathcal{D}])$. Then, one can extract the decryption request message $(\tau, \mathcal{P}, C, [\mathsf{p}])$ and check if $[\mathsf{p}] = [\mathsf{p}']$ and verify that partial decryptions of $[\mathsf{p}']$ indeed recover p.

4.2 Threshold SHE-based Instantiation

TFHE provides the capability to compute on encrypted data without interacting with other parties. Unfortunately, FHE schemes have high performance overhead in practice and we are not aware of an implementation of the recently proposed TFHE schemes [32, 1]. To this end, we describe a scheme based on somewhat homomorphic encryption scheme which is more efficient in practice but supports only a small subset of the functionality provided by FHE. This subset, however, can be extended through interaction between the parties. See Figure 3 for an illustration of this scheme.

Somewhat-Homomorphic Encryption. Somewhat homomorphic encryption (SHE) enables to evaluate a subset of computable functions (e.g., either only additions or multiplications). For example, the BGN scheme [6] supports additions but only one multiplication. Unfortunately, BGN requires one to compute discrete logarithm during decryption which puts an upper bound on the plaintext space which is too restrictive for reasonable datasets. Instead, we choose to base our implementation on the Paillier Cryptosystem [42] that is only additively-homomorphic meaning that addition and scalar multiplication are computable but multiplication of ciphertexts is not supported. The primary feature of the scheme is that it supports a large message space with efficient decryption which proves essential for our work (detailed in Section 5). Nonetheless, by using interactive protocols, it is possible to "upgrade" Paillier to also support homomorphic multiplication. Our work relies on the threshold variant of the scheme described in [15] where parties receive shares of the secret key as in the TFHE case. We refer the reader to [15] for details of the scheme.

In the rest of the section we abstract threshold-SHE (TSHE) as a scheme that allows some computations to be done on ciphertexts locally, by using its limited homomorphic properties. For other operations (e.g., multiplication) we rely on interaction with other parties where a majority is assumed to be online. Similar to TFHE, it also requires a majority of the parties to decrypt the result.

TSHE HypoCert. The setup is the same as for the TFHE-based approach in the previous section except that underlying

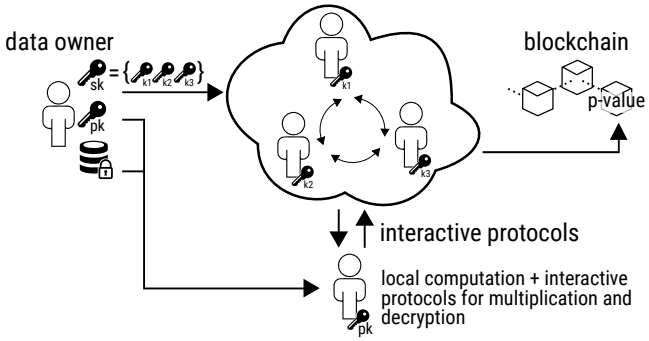


Figure 3: Overview of the threshold Paillier based HypoCert instantiation (Section 4.2). The researcher receives an encrypted dataset from the data owner and proceeds to compute statistical tests using both local computations and interactive protocols, using parties in the HypoCert network for computing non-linear functions and to certify the obtained p-value(s) on the blockchain Log.

scheme is based on TSHE. The execution of the statistical test computations, however, is different from TFHE in that interactive protocols are used for intermediate computations. These protocols are randomized and hence their results need to be recorded on the Log to ensure audit-ability.

$\mathcal{P}$ runs test C up until the computation that requires an interactive protocol. Let $\mathsf{op}_0(\mathcal{C}_0)$ denote this computation where op_0 is the operation (or gate, if using circuit notation) and $\mathcal{C}_0$ is the list of ciphertexts that $\mathcal{P}$ obtained from local computations or from the original dataset $[\mathcal{D}]$. For example, op_0 can be a multiplication and $\mathcal{C}_0$ contain ciphertexts $[a]$ and $[b]$. $\mathcal{P}$ posts to Log the message tuple $(\tau, 0, \mathsf{op}_0, \mathcal{C}_0)$. Every other party receives this message and all the parties engage in the interactive protocol to compute $\mathsf{op}_0(\mathcal{C}_0)$ and obtain the encrypted result of this computation, call it $[\mathsf{c}]$. Every party then posts a signed message tuple $(\tau, 0, [\mathsf{c}])$ to Log. $\mathcal{P}$ then uses $[\mathsf{c}]$ and proceeds with the computation. In general, for the ith interactive protocol required during the computation, $\mathcal{P}$ posts a message $(\tau, i, \mathsf{op}_i, \mathcal{C}_i)$ and the parties engage in the protocol, posting the result to Log.

Once the computation is done, $\mathcal{P}$ requests the decryption of the result $[\mathsf{p}]$ as in the TFHE case.

Security. The arguments for why confidentiality, access control and transparency properties hold for TSHE are similar to those in the TFHE instantiation: the decryption protocol is decentralized and requires a majority of the parties to participate, while recording this activity to Log. During an audit of $(\tau, \mathcal{P}, C, \mathsf{p})$, the auditor executes test C till the first operation that SHE does not support. Let op'_0 denote the first such operation and $\mathcal{C}'_0$ denote its ciphertext list. The auditor retrieves $(\tau, 0, \mathsf{op}_0, \mathcal{C}_0)$ from Log and verifies if $\mathsf{op}' = \mathsf{op}$ and $\mathcal{C}'_0 = \mathcal{C}_0$. It then gathers the transcript from the interactive protocol of op_0 to reconstruct $[\mathsf{c}]$ and proceeds with local execution of C. It continues these verification steps for every operation in C not supported by SHE.

4.3 Secret Sharing Instantiation

Our final instantiation requires more interaction than the previous two designs while simultaneously requiring that each HypoCert's party stores a share of the dataset. Compared

to the TSHE-based instantiation, parties are required to interact during *every* non-linear operation of a statistical test. However, as we will see in practice, secret sharing schemes can be more efficient as they rely on light-weight local computations compared to that of SHE. An illustration of this approach is given in Figure 4.

Linear Secret Sharing (LSS). Linear secret sharing is a technique for distributing a secret value amongst a set of parties such that a certain number of parties are needed to recover the value. Such schemes also support computation on secret-shared values through interactive protocols between the parties. A result of such computations can be also secret-shared.

Arguably the most popular LSS scheme was proposed by Shamir [43]. Shamir secret sharing is defined over a finite field $\mathbb{F}_p$ of prime order p. To share a secret $s \in \mathbb{F}_p$, a trusted dealer generates a polynomial $g(X)$ of degree t with random coefficients chosen from $\mathbb{F}_p$ such that $g(0) = s$. Each party $\mathcal{P}_i$ receives a *share* of s equal to the value of g evaluated at a point X_i. This allows for any subset of $t+1$ parties to recover the secret simply by reconstructing the polynomial g using Lagrange interpolation since g has degree t. For a more thorough technical description of Shamir secret sharing, we refer the reader to [4]. The two protocols needed for the purpose of describing the secret-shared construction of HypoCert are the following. Given a secret s, the `CreateShares`(s) protocol creates n shares which are then distributed amongst the parties accordingly. We denote a secret share of s that belongs to party i as $[\![s]\!]_i$. Then, given a set of shares of s from $t+1$ (or more) parties, `Reveal`($[\![s]\!]$) protocol reconstructs the polynomial g by interpolating the shares and outputs $s = g(0)$.

Linear operations on secret shares can be computed locally without interaction between parties. Specifically, given secret shares $[\![a]\!]$ and $[\![b]\!]$ with corresponding polynomials g and g' of degree t, it holds that $(a+b) = (g+g')(0)$ and $ac = (cg)(0)$ for some public scalar $c \in \mathbb{F}_p$. Therefore, addition and scalar multiplication can be trivially computed over secret shares. This forms a basis for more complex protocols such as multiplication of secret shares. Hence, LSS supports any computation that can be represented as a circuit. In Section 5 we present in detail the protocols that we require for statistical tests and how to instantiate them in LSS.

LSS HypoCert. During the setup the owner runs `CreateShares`() on each value of the dataset $\mathcal{D}$ amongst the set of parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ such that a majority of the parties can collectively recover the entire dataset. The owner then goes offline.

For every statistical test $\mathcal{P}_i$ wants to execute, it posts a corresponding arithmetic circuit of this test, C, (or its hash to Log. Let τ be the index of this test in Log. All the parties download the circuit, store the index of the test and engage in the computation where every multiplication involves an interactive protocol. Once the parties have obtained their share of the result $[\![\mathtt{p}]\!]_i$, they post message $(\tau, [\![\mathtt{p}]\!]_i)$ to Log. The parties then collectively reconstruct the output and $\mathcal{P}$ posts test certificate $(\tau, \mathcal{P}, C, \mathtt{p})$ to Log.

Security. The first three security properties follow from the arguments for TFHE, again since the majority of the parties is required in order to reconstruct any secret shared value.

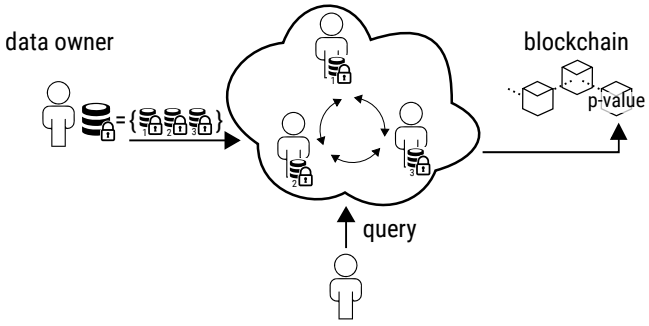


Figure 4: Overview of the secret shared based instantiation (Section 4.3). The researcher issues a query for a statistical test which is then computed by parties in HypoCert and the p-value certified. We note that no computation is performed by the researcher in this instantiation unless the researcher is one of the parties in HypoCert.

The auditing procedure is slightly different since the auditor does not have access to $\mathcal{D}$'s shares (recall that in previous two instantiations the auditor was given access to $[\mathcal{D}]$). However, since every operation has to be executed by $\mathcal{P}_j$ on its share of $\mathcal{D}$ and every party posts its share of the final result on Log, the auditor can simply verify that these messages recover the same result as the one in the certificate.

5. HypoCert BUILDING BLOCKS

Computing statistical tests such as Student's T-test, Pearson Correlation, and Chi-Squared in HypoCert requires the ability to evaluate addition, multiplication and division gates without revealing the inputs. When using non-FHE based solutions (either SHE or LSS), it is necessary to invoke *interactive protocols* to compute non-linear arithmetic gates. Furthermore, many statistical tests require computations be done over the reals which introduces additional complications when representing (and computing with) non-integer values. In this section we aim to elaborate on these details.

5.1 Statistical Security

We make extensive use of statistically[1] secure multi-party protocols presented in [9]. The advantage of using statistical security (as opposed to perfect) is that many existing protocols can be rendered far more efficient with only slightly more relaxed notions of security. More formally, consider two random variables X and Y with finite sample spaces U and V. The statistical distance between X and Y is defined to be $\Delta(X,Y) = \frac{1}{2}\sum_{w \in U \cup V} |\Pr[X = w] - \Pr[Y = w]|$. The distributions are *perfectly indistinguishable* if $\Delta(X,Y) = 0$ (i.e., as in LSS schemes) and *statistically indistinguishable* if $\Delta(X,Y)$ is negligible in a security parameter κ. We refer the reader to [8] for a full description of statistically secure building blocks as well the proof of security.

5.2 Computing Non-linear Arithmetic Gates

In Section 4 we mentioned that both SHE- and LSS-based schemes require interactive protocols for computing non-linear arithmetic gates. Since almost all statistical tests require non-linear computations (i.e., multiplications), we use existing protocols described in [22, 9, 14] which we

[1]Not to be confused with statistical tests.

Table 1: Pros and cons of the three HypoCert designs presented in Section 4.

HypoCert Design	Pros	Cons
Threshold-FHE	Minimal interaction between parties.	High performance overhead in practice.
Threshold-SHE	More efficient local computation compared to Threshold-FHE.	Requires more interaction between parties compared to Thresold-FHE.
Secret Sharing	More efficient computation (better round complexity) compared to above encryption-based instantiations.	Each party is required to store a share of the dataset and be online for *all* arithmetic operations.

summarize in this section. Note that while the protocols used in this work were originally described for the LSS setting, all protocols can be applied directly to a threshold-SHE setting, even when active security is required [13].

For the sake of visual simplicity, arithmetic additions, subtractions, and scalar multiplications are left implicit unless otherwise stated, i.e., $[\![a]\!] + [\![b]\!]$, denotes the addition of two shares and $[\![a]\!]c$ denotes the multiplication of a share [ciphertext] by a public scalar. If a value is public, we let $a + [\![b]\!]$ represent the addition of a "dummy" share $[\![a]\!]$ of the publicly known quantity a with secret share $[\![b]\!]$. In the complexity analysis, the term *invocation* is used to describe the number of calls to constant-round interactive sub-protocols (e.g., `Reveal`) which provides an upper bound on the round complexity of each described protocol.

`Mult`$([\![a]\!], [\![b]\!])$. Given secret shares $[\![a]\!]$ and $[\![b]\!]$, obtain the secret share $[\![ab]\!]$ in 1 round and 1 invocation.

`TruncPR`$([\![a]\!], \ell, m)$. Given secret share $[\![a]\!]$ and an integer m, obtain the share $[\![\tilde{a}]\!] = [\![a/2^m]\!]$. The protocol is statistically secure, probabilistically correct, and has a total complexity of 2 rounds and $2m$ invocations. While the probabilistically correct variant is sufficient for many applications, if more precise computations are required, [9] proposes a less efficient but deterministic variant of the protocol.

`BitDec`$([\![a]\!], k)$. Given secret share $[\![a]\!]$, obtain the bitwise shares $[\![a]\!]^B$ of the first k bits of a. The statistically secure variant of this protocol has a total complexity of $\log_2(k)$ rounds and $k \log_2(k)$ invocations while the constant-rounds variant has a complexity of 114 rounds and $110k \log_2(k) + 118k$ invocations.

`BitLT`$([\![a]\!]^B, [\![b]\!]^B)$. Given bitwise secret shares $[\![a]\!]^B$ and $[\![b]\!]^B$, obtain the share $[\![c]\!]$ such that $c = 1$ if $a < b$ and $c = 0$ otherwise. This protocol has a total complexity of 19 rounds and 22ℓ invocations of `Mult`. Note that we use `BitLT`$([\![a]\!]^B, b)$ to describe bitwise comparison between a bitwise share $[\![a]\!]^B$ and a public b. This can be trivially achieved by converting b to a "dummy" bitwise share $[\![b]\!]^B$ and running the comparison protocol on inputs $[\![a]\!]^B$ and $[\![b]\!]^B$.

`FPDiv`$([\![a]\!], [\![b]\!], \ell, f)$. Given secret shares $[\![a]\!]$ and $[\![b]\!]$ obtain the secret share $[\![a/b]\!]$ with f-bits of resulting precision. This protocol is based on the Goldschmidt method for achieving integer division presented in [26]. The idea behind Goldschmidt's method is to obtain initial approximations to both the divisor and dividend and iteratively compute *quadratically converging* approximations up to a desired precision. This protocol is described in [9], is statistically secure, and has a total complexity of $3 \log_2(\ell) + 2$ rounds and $1.5\ell \log_2(\ell) + 4\ell - 2$ invocations.

5.3 Fixed-Point Arithmetic

Representation of real numbers in both Paillier and secret sharing schemes is a common problem given the message space consists of elements from $\mathbb{Z}_N$ and $\mathbb{F}_p$, respectively. Performing arithmetic over real numbers is therefore non-trivial and requires either floating-point or fixed-point encoding. While floating-point representation provides better precision, it is also far less efficient compared to fixed-point representation, at least in an MPC context.

Encoding. Given a public fixed-point precision parameter f denoting bits of precision required per computation, we can approximate any real number $a \in \mathbb{R}$ as $a \approx \lfloor a2^f \rfloor 2^{-f}$. We define $\mathtt{fp}_f(x) = \lfloor a2^f \rfloor$ to be the function encoding real numbers into the corresponding fixed-point representation.

Addition/Subtraction. Addition (and subtraction) of two fixed-point numbers is trivial and can be computed without interaction. For $a, b \in \mathbb{R}$, let $\tilde{a}, \tilde{b}$ be the fixed point representations of a and b, respectively. Then, $\tilde{a} + \tilde{b} \approx (\tilde{a} + \tilde{b})2^{-f}$ with f bits of precision, as required.

Multiplication. Multiplication of fixed-point numbers is more involved as we need to *scale down* the result to have the correct f bits of precision. For $a, b \in \mathbb{R}$, let $\tilde{a}, \tilde{b}$ be the fixed point representations.

Observe that $ab \approx \tilde{a}\tilde{b}2^{-2f}$ but has $2f$ bits of precision. Hence, we need to *scale down* the product $\tilde{a}\tilde{b}$ by 2^f in order to obtain the correct precision. This can be achieved interactively using the `TruncPR` to obtain $(\tilde{a}\tilde{b})/2^f$ so that the resulting value has f bits of precision as required.

Signed Encoding. As with fixed-point encoding, there is no "natural" way to represent negative values. We designate a range of length ω (e.g., $\omega = N, p$) within $\mathbb{Z}_N$, $\mathbb{F}_p$ for the representation of negative integers. More concretely we let the range $[0, \lceil \frac{\omega}{2} \rceil)$ represent the *positive* integers in the range $[0, \lceil \frac{\omega}{2} \rceil)$ and elements in the range $[\lceil \frac{\omega}{2} \rceil, \omega)$ encode *negative* integers in the range $[-\lceil \frac{\omega}{2} \rceil, 0)$. Note that the correctness of both addition and multiplication is preserved with such encoding allowing for efficient representation and arithmetic of signed integer values in $\mathbb{Z}_N$ and $\mathbb{F}_p$.

6. STATISTICAL TESTS

Our implementation of HypoCert supports three statistical tests: Student's T-test, Pearson Correlation and Chi-Squared. While not exhaustive, this set of tests is widely used in quantitative analysis and covers common use cases where analysts want to either reason about population means, correlations between variables and differences between sets of categorical data. HypoCert is extendable and we plan to add other tests in the future.

In this section we provide background on these tests and describe how they are realized within HypoCert. Note that for simplicity we restrict ourselves to describe how the test statistics of Student's T-test, Pearson Correlation and Chi-Squared, t, r and χ^2, respectively, are computed. Converting these statistics to p-values in order to assess statistical significance can be done trivially outside of HypoCert.

6.1 Dataset Characteristics and Notation

We describe statistical tests over a dataset $\mathcal{D}$. Though we use notation of the TFHE based instantiation from Section 4.1 to denote encoded data, the tests can be executed using both TSHE and LSS instantiations since the interactive protocol descriptions are equivalent regardless of the underlying scheme [13]. For example, $\texttt{Mult}([a],[b])$ would be substituted with a multiplication over secret shared values for LSS-based instantiations.

The encrypted form of $\mathcal{D}$ is denoted by $[\mathcal{D}]$ where each entry is encrypted. We let k denote the *number of rows* in $\mathcal{D}$ and let m represent the number of columns (attributes). Therefore, $\mathcal{D}$ can be seen as a $k \times m$ matrix containing real values and $[\mathcal{D}]$ as a $k \times m$ matrix containing encrypted fixed-point approximations to the real values of entries in $\mathcal{D}$. We assume, *wlog*, that the tests are computed over the first w (where $w \geq 2$) attributes $\pi_1, \ldots, \pi_w$ of $\mathcal{D}$. As such, we denote the value of the i^{th} row of attribute j as $\mathcal{D}_{i,\pi_j}$, equivalently denoted as $[\mathcal{D}_{i,\pi_j}]$ in the encrypted dataset.

In order to allow researchers to form a hypothesis about $\mathcal{D}$, we require the data owner to release *attribute metadata* (e.g., number of attributes, their type and domain size, independence from other attributes, etc). Additionally we do not attempt to hide the size of the dataset.

6.2 Student's Student's T-test

Student's Student's T-test is used to compare means of two independent samples where the null hypothesis stipulates that there is no statistically significant difference between the two distributions [45].

Let π_1 and π_2 be two independent attributes (columns) in $\mathcal{D}$. Let x and y represent the column values of π_1 and π_2 in $\mathcal{D}$, respectively. That is, x is $\mathcal{D}_{1,\pi_1}, \ldots, \mathcal{D}_{k,\pi_1}$ and y is $\mathcal{D}_{1,\pi_2}, \ldots, \mathcal{D}_{k,\pi_2}$. Denote the mean of x and y as $\bar{x}$ and $\bar{y}$, respectively. Let the standard deviation of x and y be denoted as s_x and s_y.

The t statistic is computed according to:

$$t = \frac{|\bar{x} - \bar{y}|}{s_p \sqrt{\frac{2}{k}}} \tag{1}$$

Where $s_p = \sqrt{\frac{(k-1)s_x^2 + (k-1)s_y^2}{2k-2}}$ is an estimator of the pooled standard deviation of the two samples.

Student's T-test in HypoCert. Pseudo-code for the test is presented in Protocol 1. The protocol closely follows

Protocol 1: $t \leftarrow \texttt{StudentTTest}([\mathcal{D}], k, \pi_1, \pi_2)$

1 $([x],[y]) \leftarrow (\sum_{i=1}^{k}[\mathcal{D}_{i,\pi_1}], \sum_{i=1}^{k}[\mathcal{D}_{i,\pi_2}])$;
2 $[\bar{x}] \leftarrow \texttt{TruncPR}([x]\texttt{fp}_f(1/k), 2\ell, f)$;
3 $[\bar{y}] \leftarrow \texttt{TruncPR}([y]\texttt{fp}_f(1/k), 2\ell, f)$;
4 **foreach** $i \leftarrow 1, 2, \ldots, k$ **do parallel**
5 $\quad [d_{x_i}] \leftarrow [\mathcal{D}_{i,\pi_1}] - [\bar{x}]$;
6 $\quad [d_{y_i}] \leftarrow [\mathcal{D}_{i,\pi_2}] - [\bar{y}]$;
7 $\quad [h_{x_i}] \leftarrow \texttt{Mult}([d_{x_i}], [d_{x_i}])$;
8 $\quad [h_{y_i}] \leftarrow \texttt{Mult}([d_{y_i}], [d_{y_i}])$;
9 $([h_x],[h_y]) \leftarrow (\sum_{i=1}^{k}[h_{x_i}], \sum_{i=1}^{k}[h_{y_i}])$;
10 $[s_x^2] \leftarrow \texttt{TruncPR}([h_x]\texttt{fp}_f(1/(k-1)), 4\ell, 2f)$;
11 $[s_y^2] \leftarrow \texttt{TruncPR}([h_y]\texttt{fp}_f(1/(k-1)), 4\ell, 2f)$;
12 $[t_x] \leftarrow (k-1)[s_x^2]$;
13 $[t_y] \leftarrow (k-1)[s_y^2]$;
14 $[u] \leftarrow [t_x] + [t_y]$;
15 $[a] \leftarrow [\bar{x}] - [\bar{y}]$;
16 $[a^2] \leftarrow \texttt{Mult}([a],[a])$;
17 $[a^2] \leftarrow \texttt{TruncPR}([a^2], 2\ell, f)$;
18 $[b] \leftarrow \texttt{TruncPR}([u]\texttt{fp}_f(1/(k^2-k)), 2\ell, f)$;
19 $[t^2] \leftarrow \texttt{FPDiv}([a^2],[b])$;
20 $t^2 \leftarrow \texttt{Reveal}([t^2])$;
21 $t \leftarrow \sqrt{t^2}$;
22 **return** t;

Equation 1 while adjusting the precision of fixed-point computations using `TruncPR` as explained in Section 5. It avoids computing the square root of secret values and instead leaves it as a final operation to be computed on plaintext. We now briefly describe the steps of the protocol: Lines 1 to 3 compute the mean of the two variables. Lines 4 to 11 compute the squared standard deviation of the two variables. Lines 12 to 14 compute the pooled standard deviation of the variables s_p. Lines 15 to 17 compute the square of the numerator in Equation 1. Line 18 computes the square of the denominator in Equation 1. Finally, 21 computes the t value in the clear by taking the square root of the revealed t^2 value.

Requirements. Let $\eta \in \mathbb{N}$ be an upper bound on the largest absolute value in $\mathcal{D}$ (i.e., given as part of attributes' domain size), to evaluate a Student's T-test over any two attributes $\mathcal{D}$, the following constraints must hold to ensure no "overflow" occurs during computation: $\ell > 4\log_2(\eta) + 3\log_2(k) + f$ and $N, p > 2^{4\ell+\kappa+\nu+1}$ where $\nu = \log_2(n)$.

Complexity. All sub-protocols invoked are constant-rounds with the exception of `FPDiv` which requires a number of rounds proportional to $\log_2(\ell)$. Therefore, the complexity of the protocol hinges on the `FPDiv` invocation making the final complexity $O(\log_2(\ell))$ rounds and 11 invocations.

6.3 Pearson's Correlation Test

Pearson Correlation test is used to compare the linear correlation between two continuous independent variables. The result r lies in the range $[-1, 1]$ corresponding to negative or positive correlation level between the variables [45].

Let π_1 and π_2 be two continuous independent attributes.

Let variables x and y represent the values of π_1 and π_2 where $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ denote the observed values for x and y, respectively. Denote the mean of x and y as $\bar{x}$ and $\bar{y}$. Pearson Correlation's correlation coefficient r is then computed according to the following equation:

$$r = \frac{\sum_{i=1}^{k}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{k}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{k}(y_i - \bar{y})^2}} \quad (2)$$

Pearson Correlation in HypoCert. Pseudo-code for computing Pearson Correlation coefficient is presented in Protocol 2. The protocol closely follows Equation 2 to compute the statistic and uses `TruncPR` to adjust the precision of fixed-point values as explained in Section 5. Lines 1 to 3 compute the mean of the two variables. Lines 4 to 12 compute the sum of the squared deviations of the two variables and the sum of the products of the deviations. Lines 13 to 16 compute the square of the numerator. Lines 17 to 18 compute the square of the denominator. Lines 19 to 22 extract the sign of r^2. Finally, 24 computes the r value in the clear by taking the square root of the revealed r^2 value and using the sign bit to set the sign of r.

Protocol 2: $r \leftarrow$ `PearsonTest`$([\mathcal{D}], k, \pi_1, \pi_2)$

1 $([x], [y]) \leftarrow (\sum_{i=1}^{k}[\mathcal{D}_{i,\pi_1}], \sum_{i=1}^{k}[\mathcal{D}_{i,\pi_2}])$;
2 $[\bar{x}] \leftarrow$ `TruncPR`$([x]\texttt{fp}_f(1/k), 2\ell, f)$;
3 $[\bar{y}] \leftarrow$ `TruncPR`$([y]\texttt{fp}_f(1/k), 2\ell, f)$;
4 **foreach** $i \leftarrow 1, 2, \ldots, k$ **do parallel**
5 $\quad [d_{x_i}] \leftarrow [\mathcal{D}_{i,\pi_1}] - [\bar{x}]$;
6 $\quad [d_{y_i}] \leftarrow [\mathcal{D}_{i,\pi_2}] - [\bar{y}]$;
7 $\quad [d^2_{x_i}] \leftarrow$ `Mult`$([d_{x_i}], [d_{x_i}])$;
8 $\quad [d^2_{y_i}] \leftarrow$ `Mult`$([d_{y_i}], [d_{y_i}])$;
9 $\quad [e_i] \leftarrow$ `Mult`$([d_{x_i}], [d_{y_i}])$;
10 $([s_x], [s_y]) \leftarrow (\sum_{i=1}^{k}[d^2_{x_i}], \sum_{i=1}^{k}[d^2_{y_i}])$;
11 $[s_x] \leftarrow$ `TruncPR`$([s_x], 2\ell, f)$;
12 $[s_y] \leftarrow$ `TruncPR`$([s_y], 2\ell, f)$;
13 $[a] \leftarrow \sum_{i=1}^{k}[e_i]$;
14 $[a] \leftarrow$ `TruncPR`$([a], 2\ell, f)$;
15 $[a^2] \leftarrow$ `Mult`$([a], [a])$;
16 $[a^2] \leftarrow$ `TruncPR`$([a^2], 2\ell, f)$;
17 $[b] \leftarrow$ `Mult`$([s_x], [s_y])$;
18 $[b] \leftarrow$ `TruncPR`$([b], 2\ell, f)$;
19 $[r^2] \leftarrow$ `FPDiv`$([a^2], [b])$;
20 $[\![a]\!]^B \leftarrow$ `BitDec`(a, ℓ);
21 $[c] \leftarrow$ `BitLT`$([\![a]\!]^B, \lceil \frac{\ell}{2} \rceil)$ `// extract sign bit`
22 $c \leftarrow$ `Reveal`$([c])$;
23 $r^2 \leftarrow$ `Reveal`$([r^2])$;
24 $r \leftarrow \sqrt{r^2} - 2c\sqrt{r^2}$;
25 **return** r;

[2]We note that the bit decomposition protocol described in [9] maps the decomposed value to the range $[0, 2^\ell)$ and therefore the sign of $[a]$ is extracted correctly even in the case when $2^\ell < N, p$ and $a > 2^\ell$ which occurs when using the proposed encoding in Section 5.

Requirements. Let $\eta \in \mathbb{N}$ be an upper bound to the largest absolute value in $\mathcal{D}$, to evaluate a Pearson Correlation test over any two attributes $\mathcal{D}$, the following requirements must hold for these user-set parameters to ensure no "overflow" occurs during computation: $\ell > 4\log_2(\eta) + 4\log_2(k) + f$ and $N, p > 2^{2\ell+\kappa+\nu+1}$.

Complexity. All sub-protocols invoked are constant-rounds with the exception of `FPDiv` and `BitDec` [3] which requires a number of rounds proportional to $\log_2(\ell)$. We therefore conclude that the protocol requires $O(\log_2(\ell))$ rounds and 17 invocations.

6.4 Chi-Squared Test

The Chi-Squared test determines whether the sampling distribution of the test statistic follows a χ^2 distribution when the null hypothesis is true[11, 45]. The Chi-Squared test is useful in determining whether there is a significant difference between the expected frequencies and the observed frequencies in a set of observations from *mutually exclusive* categories. In other words, Chi-Squared evaluates the "goodness of fit" between a set of expected values and observed values, the test result is deemed significant if the expected frequencies match the observed frequencies.

For a collection of k observations classified into w mutually exclusive categories where each observed value is denoted by x_i for $i = 1, 2, \ldots, w$, denote the probability that a value falls into the i^{th} category by α_i such that $\sum_{i=1}^{w}\alpha_i = 1$. Note that the expected value for each category is $e_i = n\alpha_i$. The Chi-Squared statistic is computed according to the following equation:

$$\chi^2 = \sum_{i=1}^{w}\frac{(x_i - e_i)^2}{e_i} \quad (3)$$

Let $\mathcal{D}$ contain w mutually exclusive attributes (categories) $\pi_1, \ldots, \pi_w$ such that $\mathcal{D}_i, \pi_j \in \{0, 1\}$ for all $i = 1 \ldots k$ and $j = 1 \ldots w$. In other words, each category is a Boolean flag representing whether a row i in $\mathcal{D}$ is in the category j. Given such a "raw" dataset $\mathcal{D}$, we need a way to convert $\mathcal{D}$ into histogram form $\mathcal{H} = (h_1, h_2, \ldots, h_w)$, filtered based on the selected categories, so as to compute the Chi-Squared statistic over $\mathcal{H}$. To achieve this in a private and secure manner, we must first "pre-process" $[\mathcal{D}]$ into a histogram $\mathcal{H}$ containing the summations of the w categories selected by the user.

Note: we use $\mathcal{H}$ for purpose of providing a general solution and to remain consistent with the descriptions of the previous two tests (i.e., $\mathcal{D}$ has the same format across all tests). If $\mathcal{D}$ is already in histogram form, the the Chi-Squared test may be applied directly on $\mathcal{D}$.

Chi-Squared in HypoCert. Pseudo-code for computing the Chi-Squared test is presented in Protocol 3. It closely follows Equation 3. The first for-loop (line 1 computes the histogram $[\mathcal{H}] = ([h_1], \ldots, [h_w])$ from $[\mathcal{D}]$. The correctness of the computed histogram $\mathcal{H}$ follows from the fact that each attribute in the set is mutually exclusive, i.e., if $\mathcal{D}_{j,\pi_i} = 1$ then $\mathcal{D}_{j,\pi_w} = 0$ for all $j = 1 \ldots k$ and $i \neq w$. Line 3 computes the sum of all the values in $[\mathcal{H}]$. The second for-loop (line 4 computes the *expected values* of each entry in the $[\mathcal{H}]$ as

[3]If the statistically secure variant of the protocol is used. A more complex, constant-round solution is presented in [14] but is less efficient in practice when ℓ is small.

well as the $(x_i - e_i)$ term of Equation 3. The third for-loop (line 7 computes each term in the summation. Finally, line 11 computes the Chi-Squared statistic by summing over all the individual terms.

Protocol 3: $\chi^2 \leftarrow \texttt{ChiSq}([\mathcal{D}], k, \{\pi_1, ..., \pi_w\}, \{\alpha_1, ..., \alpha_w\})$

1 **foreach** $i \leftarrow 1, 2, \ldots, w$ **do parallel**
2 | $[h_i] \leftarrow \sum_{j=1}^{k} [\mathcal{D}_{j,\pi_i}]$;
3 $[s] \leftarrow \sum_{i=1}^{w} [h_i]$;
4 **foreach** $i \leftarrow 1, 2, \ldots, w$ **do parallel**
5 | $[e_i] \leftarrow [s]\alpha_i$;
6 | $[d_i] \leftarrow [h_i] - [e_i]$;
7 **foreach** $i \leftarrow 1, 2, \ldots, w$ **do parallel**
8 | $[w_i] \leftarrow \texttt{Mult}([d_i], [d_i])$;
9 | $[w_i] \leftarrow \texttt{TruncPR}([w_i], 2\ell, f)$;
10 | $[x_i] \leftarrow \texttt{FPDiv}([w_i], [e_i])$;
11 $[\chi^2] \leftarrow \sum_{i=1}^{w} [x_i]$;
12 $\chi^2 \leftarrow \texttt{Reveal}([\chi^2])$;
13 **return** χ^2

Requirements. Let $\eta \in \mathbb{N}$ be an upper bound to the largest absolute value in $\mathcal{D}$. Let $\mathcal{H}$ be a histogram with w attributes ($w \geq 2$). To correctly perform the Chi-Squared test over the w selected attributes in $\mathcal{D}$, the following requirements must hold for these user-set parameters to ensure no "overflow" occurs during computation: $\ell > 2\log_2(\eta) + 3\log_2(k) + f$ and $N, p > 2^{2\ell+\kappa+\nu+1}$ where $\nu = \log_2(n)$.

Complexity. All sub-protocols invoked are constant-rounds with the exception of `FPDiv` which requires a number of rounds proportional to $\log_2(\ell)$. We therefore conclude that the protocol requires $O(w \log_2(\ell))$ rounds and 4 invocations.

7. EXPERIMENTAL EVALUATION

The primary goal of our experiments is to evaluate the practicality of HypoCert on real-world sized datasets for three common types of statistical tests. The experiments are designed to achieve three goals: 1) measure the computation time required for each test with different dataset sizes and number of participants, 2) benchmark the number of *multiplications* required for each test and combination of inputs and 3) ensure the statistics computed are accurate and fall within the required precision range with high probability (recall that `TruncPR` is probabilistically correct).

Implementation and Environment. We implemented HypoCert using the secret-shared instantiation of HypoCert (Section 4.3). Our implementation is in Go 1.10.1 and is available at `https://github.com/sachaservan/hypocert`. In order to instantiate threshold somewhat-homomorphic encryption HypoCert, we implemented the threshold-Paillier scheme and associated protocols from [13, 15] which we use for comparing with the secret-shared scheme.

All experiments are conducted on a single machine with Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz processors (40 cores in total) and 256GB of RAM, running Ubuntu 16.04 LTS. Unless otherwise stated, each result is the average over five separate trial runs (for each combination of parameters). In most cases the variance across runs was minimal, but we report 95%-confidence markers (under a normal approximation assumption).

Datasets. We generate three datasets of 1,000, 5,000, and 10,000 rows with random real values ranging between 1 to 100. Note that generating datasets at random does not advantage HypoCert in any way but gives us control over the size and complexity of the dataset across all three statistical tests. We use the same set of datasets to benchmark both Student's T-test and Pearson Correlation tests. Given the need for categorical data when computing a Chi-Squared test, we generate a total of nine separate datasets of 1,000, 5,000 and 10,000 rows and 5, 10 and 20 attributes (categories).

Experimental Setup. We benchmark all statistical tests with $3, 5, 9$ and 17 parties, where a threshold majority is required to reconstruct shares. Each party is run as a separate process and with access to two cores on the computing machine. In other words, each party is simulated as a dual-core machine, separate from other parties[4]. This setup allows us to simulate a multi-party computing environment with control over all variables, (e.g., network latency). We set the network latency between parties to be 1ms which simulates latencies observed on a local area network (LAN); a standard setup used for benchmarking MPC protocols [49, 5].

Parameters. To ensure all tests are benchmarked in a way that enables comparisons between results, we fix the parameters ahead of time to satisfy the requirements imposed by *all three* statistical tests (and datasets) and do not change the parameters between experiments. We set $f = 20$ which provides approximately 6 decimal places of precision for the Chi-Squared test and 3 decimal places for Student's T-test and Pearson Correlation test (due to the square-root computation at the end in both tests). We let the statistical security parameter $\kappa = 40$ which provides 40-bits of statistical security. The largest value in our datasets is $\eta_{max} = 100$ and the number of entries in our largest dataset is $k_{max} = 10000$. Therefore, we fix $\ell = 100$ so that $\ell > 4\log_2(\eta_{max}) + 4\log(k_{max}) + f$ which indeed satisfies the all requirements.

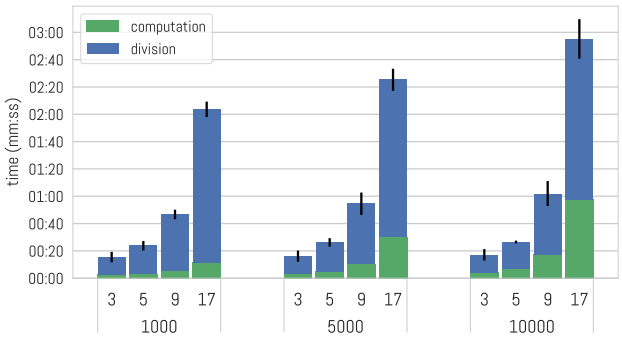
Finally, set p (the order of the secret sharing field $\mathbb{F}$) to be a random prime of at least $4\ell + \kappa + f + \log_2(n_{max}) + 1$ bits ($\approx$ 460-bits in our case), where $n_{max} = 17$ is the maximum number of parties in our experiments. This guarantees the field $\mathbb{F}_p$ is large enough for all computations performed and satisfies the statistical security requirements.

Results. Setting up the system, i.e., code executed by the data owner, was consistently below 67 seconds across all experiments (mean 4.03 seconds, std. 6.89).

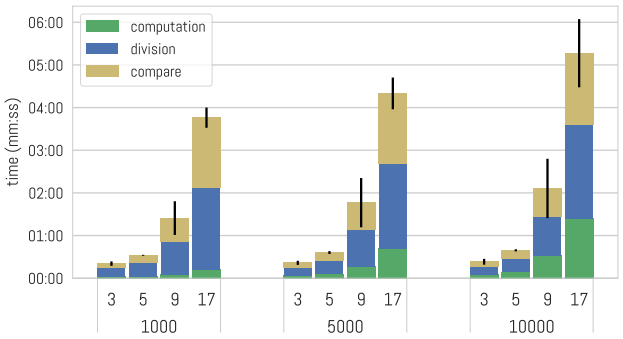
We compare the resulting precision of each statistical test computed in HypoCert with results obtained from computing the test using the Python SciPy Library[36]. The mean absolute errors for each test are reported in Table 2.

Run-time comparisons for each statistical test, dependent on the combination of parties and datasets, is reported in Figure 5. In all of our experiments, fixed-point division required the most computation time which is expected considering that it is the most complex protocol invoked by each test.

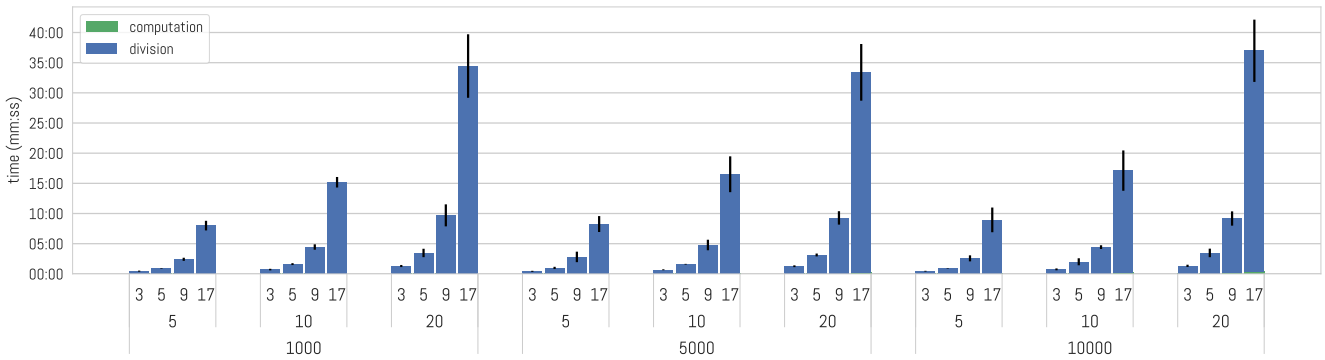
[4]The machine on which the experiments are conducted has 40 cores which allows such a setup without CPU swapping.



(a) Total run-time for computing Student's Student's T-test.



(b) Total run-time for computing Pearson Correlation test.



(c) Total run-time for computing Chi-Squared test.

Figure 5: Run-time comparisons for each of the three tests when using HypoCert based on linear secret-shared scheme (Section 4.3). The number of parties is reported below each bar. For Chi-Squared test, the number of attributes (5, 10, and 20) is reported under the number of parties. The size of the dataset is reported below each cluster of bars.

	Mean Absolute Error
Student's T-test	3.396×10^{-10}
Pearson Correlation Test	8.662×10^{-7}
Chi-Squared Test	0.0
Combined	1.733×10^{-7}
Standard deviation	4.980×10^{-7}

Table 2: Absolute error of tests computed in HypoCert.

The comparison protocol (line 21 of Protocol 2) required almost as much computation time as the division protocol (when the invocation of `BitDec` is taken into account). The effect of computing a secure division and comparison on the overall run-time is displayed in Figure 5b. The complexity of these protocols comes almost entirely from the invocation of `BitDec` which is notoriously expensive, even when the more efficient statistically-private variant is used.

Perhaps surprisingly, the number of parties involved has a larger impact on performance than the size of the dataset. This is due to the high overhead of computing division (and comparisons in the case of Protocol 2) over secret data. Both Protocol 1 and 2, require a single division operation making the computation independent of the dataset size. Therefore, this relationship does not hold true for significantly larger datasets as the number of multiplications increases linearly with the number of rows in the dataset but the number of division and comparison operations stays constant.

When computing the Chi-Squared test, division is overwhelmingly the dominant contributor to the overall run-time (see Figure 5c). This is due to the nature of the Chi-Squared statistic which requires one division *per category* (in the general case) which equates to a total of 20 calls to `FPDiv` for the largest dataset used in the Chi-Squared experiments. Under certain assumptions, it is possible to get away with doing a single division operation (for the purpose of computing the reciprocal), for example, when comparing against a uniform distribution.

We also measure the total number of multiplications required per statistical test and combination of parameters. Since `Mult` is the fundamental building block of *all* interactive protocols, this measurement provides an estimate for the number of rounds required per test. We report the results in Table 3 and 4).

Finally, we provide a run-time comparison for single invocation of `Mult` when using threshold-Paillier and LSS (as a function of the number of parties). We report these measurements in Figure 6. The reported run-time is the average over 1000 runs, for each combination of parameters.

Using Paillier appears impractical when a large number of multiplications are required (e.g., for computing secure division) or when many parties are present in the system. In practice, a single invocation of `Mult` using threshold-Paillier requires 4 rounds compared to 2 rounds when using LSS. Furthermore, the `Mult` protocol requires four exponentia-

tions (mod N^2) when threshold-Paillier is used but zero exponentiations when using LSS.

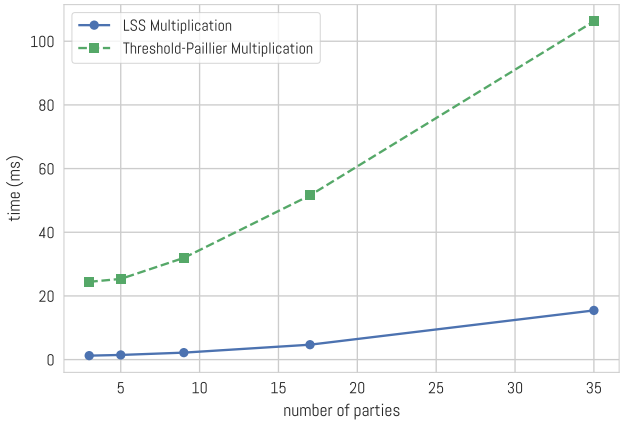


Figure 6: Run-time comparisons for a single invocation of the `Mult` protocol using threshold-Paillier and linear secret sharing with 1ms network latency.

	k	# of `Mult` invocations
Student's T-test	1,000	14,019
	5,000	22,019
	10,000	32,019
Pearson Correlation Test	1,000	27,692
	5,000	39,692
	10,000	54,692

Table 3: Number of multiplications required to compute a Student's T-test and Pearson Correlation test depending on the number of entries in the dataset.

	w	# of `Mult` invocations
Chi-Squared Test	5	59,755
	10	11,9500
	20	238,990

Table 4: Number of multiplication required to compute a Chi-Squared test depending on the number of *categories* in the dataset. Note that to compute a Chi-Squared test, the number of entries in the dataset does not contribute to multiplications.

8. RELATED WORK

Much of the related work surrounding HypoCert either focuses on private computations using FHE and MPC or non-cryptographic methods for preventing "p-hacking". To our knowledge, there has been no prior work using cryptographic techniques for ensuring validity of statistical tests.

Lauter *et al.* [39] and Zhang *et al.*[48] also propose the use of FHE to compute statistical tests on encrypted genomic data. However, the constructions are not intended for validating statistical testing procedures but rather for guarding the privacy of patient data. Lauter *et al.* make several simplifications such as not performing encrypted division (rather performing arithmetic division in the clear), imposing assumptions on the data, etc., making their solution less general compared to HypoCert. Zhang *et al.*[48] propose two methods to perform division based on a lookup table and secure approximation. FHE, SHE and LSS techniques have been also shown applicable when outsourcing computation of several machine learning tasks on private data [46, 7, 47, 27]. Our work, however, crucially relies on the decryption functionality being decentralized in order to control and keep track of data exposure.

Several techniques exist to guard against the MCP in statistical analysis. For example, there are several procedures that adjust p-values in scenarios where multiple hypotheses are examined at once. These range from conservative protocols that bound the family-wise error rate [17] to more relaxed procedures that bound the ratio of false rejections among the rejected tests (false discovery rate FDR) [3] or the marginal FDR [50, 20]. When applied properly these techniques prevent "p-hacking". However, they do not guard against intentional or unintentional bias and there is no provable way for external auditors to verify that these procedures were applied correctly. We leverage these techniques, especially incremental procedures that bound the marginal FDR [50, 20]. By operating on encrypted data and tracking and storing tests and results in tamperproof logs HypoCert can certify that MCP procedures are indeed applied as intended.

Dwork *et al.* [19, 18] propose a method that constrains analyst's access to a hold-out dataset as follows. A trusted party keeps a hold-out dataset and answers up to m hypotheses using a differentially private algorithm. Here, m depends on the size of the hold-out data and the generalization error that one is willing to tolerate. Hence, compared to our decentralized approach, this method assumes trust into a single party that (1) does not release the dataset to the researchers and (2) does not answer more than m hypotheses queries.

Finally, we note that preserving privacy of dataset values when releasing results of statistical tests [21, 34] is orthogonal to our work.

9. CONCLUSIONS

In this paper we presented HypoCert, a system that certifies hypothesis testing using proven cryptographic techniques and a decentralized certifying authority. HypoCert computes statistical test over encrypted data and uses blockchain technology to provide auditability of those results. We believe that HypoCert is a viable solution to prevent "p-hacking" and control for false-discoveries in statistical testing and a way that promotes accountability and reporducability in scientific studies. We discuss various theoretical constructions of HypoCert and provide details on our particular implementation in which we support three common statistical tests. We evaluate this implementation on various configurations and dataset sizes.

Acknowledgments

We would like to thank Anna Lysyanskaya for providing us with editorial feedback and design suggestion as well as Andy van Dam for his support.

10. REFERENCES

[1] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 483–501. Springer, 2012.

[2] C. G. Begley and L. M. Ellis. Drug development: Raise standards for preclinical cancer research. *Nature*, 483(7391):531, 2012.

[3] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300, 1995.

[4] D. Bogdanov. Foundations and properties of shamirs secret sharing scheme research seminar in cryptography. *University of Tartu, Institute of Computer Science*, 1, 2007.

[5] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[6] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference*, pages 325–341. Springer, 2005.

[7] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[8] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.

[9] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pages 35–50. Springer, 2010.

[10] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi. Solidus: Confidential distributed ledger transactions via PVORM. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 701–717, 2017.

[11] W. G. Cochran. The $\chi 2$ test of goodness of fit. *The Annals of Mathematical Statistics*, pages 315–345, 1952.

[12] A. Cockburn, C. Gutwin, and A. Dix. Hark no more: on the preregistration of chi experiments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 141. ACM, 2018.

[13] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 280–300. Springer, 2001.

[14] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.

[15] I. Damgård, M. Jurik, and J. B. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[16] K. Dickersin, S. Chan, T. Chalmersx, H. Sacks, and H. Smith Jr. Publication bias and clinical trials. *Controlled clinical trials*, 8(4):343–353, 1987.

[17] O. J. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56(293):52–64, 1961.

[18] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth. Guilt-free data reuse. *Communications of the ACM*, 60(4):86–93, 2017.

[19] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. L. Roth. Preserving statistical validity in adaptive data analysis. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 117–126. ACM, 2015.

[20] D. P. Foster and R. A. Stine. α-investing: a procedure for sequential control of expected false discoveries. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(2):429–444, 2008.

[21] M. Gaboardi, H. Lim, R. Rogers, and S. Vadhan. Differentially private chi-squared hypothesis testing: Goodness of fit and independence testing. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 2111–2120, 2016.

[22] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM, 1998.

[23] C. Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[24] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–148. Springer, 2011.

[25] M. M. Ghassemi, S. E. Richter, I. M. Eche, T. W. Chen, J. Danziger, and L. A. Celi. A data-driven approach to optimized medication dosing: a focus on heparin. *Intensive care medicine*, 40(9):1332–1339, 2014.

[26] R. E. Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.

[27] T. Graepel, K. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology (ICISC)*, 2013.

[28] M. L. Head, L. Holman, R. Lanfear, A. T. Kahn, and M. D. Jennions. The extent and consequences of p-hacking in science. *PLoS biology*, 13(3):e1002106, 2015.

[29] K. E. Henry, D. N. Hager, P. J. Pronovost, and S. Saria. A targeted real-time early warning score (trewscore) for septic shock. *Science translational medicine*, 7(299):299ra122–299ra122, 2015.

[30] J. P. Ioannidis. Contradicted and initially stronger effects in highly cited clinical research. *Jama*, 294(2):218–228, 2005.

[31] J. P. Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.

[32] A. Jain, P. M. Rasmussen, and A. Sahai. Threshold fully homomorphic encryption. *IACR Cryptology*

ePrint Archive, 2017:257, 2017.

[33] L. K. John, G. Loewenstein, and D. Prelec. Measuring the prevalence of questionable research practices with incentives for truth telling. *Psychological science*, 23(5):524–532, 2012.

[34] A. Johnson and V. Shmatikov. Privacy-preserving data exploration in genome-wide association studies. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1079–1087, 2013.

[35] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

[36] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2018-07-01].

[37] N. L. Kerr. Harking: Hypothesizing after the results are known. *Personality and Social Psychology Review*, 2(3):196–217, 1998.

[38] L. Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[39] K. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. In *International Conference on Cryptology and Information Security in Latin America*, pages 3–27. Springer, 2014.

[40] L. Mayaud, P. S. Lai, G. D. Clifford, L. Tarassenko, L. A. G. Celi, and D. Annane. Dynamic data during hypotensive episode improves mortality predictions among patients with sepsis and hypotension. *Critical care medicine*, 41(4):954, 2013.

[41] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. http://www.bitcoin.org/bitcoin.pdf.

[42] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[43] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[44] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 24–43. Springer, 2010.

[45] T. H. Wonnacott and R. J. Wonnacott. *Introductory statistics*, volume 5. Wiley New York, 1990.

[46] D. Wu and J. Haven. Using homomorphic encryption for large scale statistical analysis. Technical report, Technical Report: cs. stanford. edu/people/dwu4/papers/FHESI Report. pdf, 2012.

[47] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. E. Lauter, and M. Naehrig. Crypto-nets: Neural networks over encrypted data. *CoRR*, abs/1412.6181, 2014.

[48] Y. Zhang, W. Dai, X. Jiang, H. Xiong, and S. Wang. Foresee: Fully outsourced secure genome study based on homomorphic encryption. 15:S5, 12 2015.

[49] Y. Zhang, A. Steele, and M. Blanton. Picco: a general-purpose compiler for private distributed computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 813–826. ACM, 2013.

[50] Z. Zhao, L. De Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska. Controlling false discoveries during interactive data exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 527–540. ACM, 2017.