Proof Sketches: Verifiable Multi-Party Aggregation

REMEmBER TO ANONYMIZE
Minos Garofalakis, Joseph M. Hellerstein, Petros Maniatis
Intel Research Berkeley and UC Berkeley

ABSTRACT
Recent work on distributed aggregation has assumed a benign population of participants. In most modern distributed systems, it is now necessary to account for adversarial behavior. In this paper we consider the problem of ensuring verifiable yet efficient results to typical aggregation queries in a distributed, multi-party setting. We describe a general framework for the problem, including the threat model for adversaries that we consider. We then present a mechanism called a proof sketch, which uses a compact combination of cryptographic signatures and Flajolet-Martin sketches to verify that a query answer is within acceptable error bounds with high probability. When verification fails, we provide efficient mechanisms to identify any participants responsible for the perturbed result. We derive proof sketches for a broad class of aggregation functions, including counts, distinct counts, distinct sampling, and frequency moment estimation, and describe the characteristics of aggregation functions that are amenable to proof sketches. Finally, we examine the practical use of proof sketches, measuring their space overhead, and tempering our worst-case bounds for detecting misbehavior: we observe that adversaries can often be caught on smaller violations in practice.

1. INTRODUCTION
In recent years, distributed query processing has been a topic of interest in a number of settings, including network and distributed system monitoring, sensor networks, peer-to-peer systems, data integration systems, and web services. Many of these environments depend upon the participation of multiple parties with varying degrees of mutual trust. An outstanding research challenge is to provide trustworthy query results in environments with mutually distrustful or even adversarial parties. Note that even within a single organization like a corporation, viruses and “bot nets” make the presence of adversarial nodes a reality of modern computing that needs to be addressed. The lack of trust in such settings is a significant impediment to the adoption of distributed query technologies.

Trustworthy multiparty query processing is a problem with many facets. In this paper we consider one basic building block: ensuring verifiable but efficient computation of distributed multi-party aggregation queries. We focus particularly on in-network aggregation, in which the query processing is pushed down into the network and executed in a distributed fashion by multiple participants. In-network processing is important in settings where centralized data warehousing is undesirable, either due to technical considerations like high data rates (as in network packet monitoring), or to administrative concerns of policy and/or provisioning cost (as in peer-to-peer systems).

The challenge we consider in this paper is to partition the aggregate processing among the participants, without allowing faulty or even malicious parties to undetectably perturb the correct computation and delivery of the result. Recent work has addressed the problem of communication faults in this setting [1, 6–8], but these techniques remain vulnerable to parties that tamper with the computation. Such malicious misbehavior includes manufacturing spurious data that was not provided by authentic data generators, and suppression of data from the data generators; such activities can perturb aggregate results arbitrarily.

In this paper we develop a class of certificate called a proof sketch that allows parties in a distributed aggregate computation to prove that the final result could not have been perturbed by more than a small error bound with high probability. The proof sketches we propose achieve this result by combining the popular Flajolet-Martin (FM) sketch technique [3] with compact signatures we call authentication manifests. Our proof sketches are typically logarithmic in the size of the distributed data, and can be computed, communicated, and verified in logarithmic time. Given a proof sketch, a verifier can check that the final aggregate was, with high probability, constructed solely by merging data from the data generators. With the help of some compact additional knowledge and one more proof sketch, the verifier can check that none of the input data was suppressed. If both these checks pass, the verifier can guarantee that the aggregate result is within desired error bounds with a given confidence. When a check fails, efficient means are provided to detect participants (if any) who tampered with the computation. 

A proof sketch can either accompany complete aggregate results as a certificate, or can be used standalone to provide verifiable approximate query results. We derive a family of FM-based proof sketches.
for a broad class of aggregation functions, including counts, distinct counts, distinct sampling, and frequency moment estimation. [JMH: Ensure this is accurate?] However, we do not rely on FM sketching per se – we provide a general characterization of the key requirements of a sketch that could allow it to be used as a proof sketch.

We evaluate proof sketches empirically along two dimensions. First, we examine their space overhead in a practical setting to ensure that the constants involved are acceptable. Second, we temper our theoretical worst-case analyses of detection, and show that in practice we can often detect even smaller-scale tampering than our theoretical bounds can guarantee.

1.1 Structure of the Paper

In Section 2, we begin with a careful definition of the distributed verifiable aggregation problem in terms of a threat model that includes two basic forms of misbehavior: inaccurate reporting of base data, and incorrect computation of intermediate aggregate results. We demonstrate the problem and our approach with a simplified example, a verifiable union aggregate in Section 2.2. We introduce proof sketches in Section 3 and use them to tackle verifiable probabilistic counting (Section 3), verifiable distinct sampling (Section 4), and – combining the two – verifiable frequency moment estimation (Section 5). In Section 8 we address several extensions to basic proof sketches for misbehavior detection, and for early commitment. We then conclude with related and future work.

2. PROBLEM SETTING

In this section, we introduce a motivating scenario drawn from distributed anomaly detection, to place into context the problem of distributed verifiable aggregation. We then describe the aggregation functionality with respect to this scenario, and outline our threat model.

2.1 Motivating Scenario

Throughout our discussion, we will consider a simple example scenario in corporate asset monitoring. It is common practice for most computers owned by a corporation to run a local host intrusion detection (HID) agent such as BlackICE or SNORT. We consider such an environment, with HIDs generating events of the sort “I am under a NIMDA attack,” or “Host a.b.c.d probed my unused TCP port x.” Periodically, each HID generates summary events, such as “24 distinct hosts probed unused TCP ports in the last 24 hours.”

The querier in our scenario is a network security engineer working at corporate IT headquarters, who poses distributed aggregate queries to the collection of HIDs, to understand the features of a suspected emerging distributed worm or virus. One typical query might be “how many HIDs identified exploit X?” (a counting query). We refer to this as a predicate poll of the HIDs, since it essentially counts the predicate’s “yes votes” among the nodes in the system. Another typical query would be “return a property (e.g. the OS version number) of $k$ randomly chosen HIDs that identified exploit X” (a sampling query). In both these scenarios, each HID needs to contribute at most one data record to the aggregation; this is natural for minimizing bandwidth requirements and latency in these time-sensitive queries. However, one can also imagine queries for which each HID produces multiple matches, e.g., “count all firewall log entries that identify an instance of exploit X”.

Note that while the end systems generating data may belong to one company, the same might not be true of the aggregation infrastructure used. For instance, the aggregation functionality may be hosted at appropriate points on the Internet by a third party – e.g., a content distribution vendor like Akamai – using “edge server includes” [2] or similar technologies. As another example, multiple organizations may want to pool their network resources to compute the aggregates globally. In this case, the end users posing questions may require extra assurances that the received aggregation results reflect accurately what the trusted data sources (the HIDs) produced.

In the remainder of this paper, we assume that the participating nodes in the system are connected, and that queries from end users can be disseminated to all the relevant agents in the computation, including data generators and aggregators. The details of how the query dissemination is performed are not the topic of this paper, but are treated extensively in the literature [5, 6, 10, 13]. In some environments, queries can be disseminated along with the HID software (e.g., every HID reports on a predetermined set of predicate polls or other queries periodically). Alternatively, queries can be cryptographically signed by the querier to preserve integrity, and disseminated to the relevant agents via the (untrusted) network infrastructure (e.g., “the queries for the next hour are $X$, $Y$, and $Z$”). Queries that fail to reach all agents for whatever reason will cause our verification techniques developed below to raise alerts at the querier. We return to the issue of failed dissemination in Section 2.2. [JMH: Don’t forget we’ll need a discussion section on disconnected/dead nodes, and how in practice you distinguish that from a (selective) suppression attack.]

2.2 Aggregation Functionality

In our discussion, we adopt the distributed aggregation terminology of TAG [6]. An aggregate function like SUM, MIN, MAX, or AVERAGE is implemented via three functions: an initializer $I$, a merging function $M$, and an evaluator function $E$. In general, $M$ has the following structure:

$$< z = M(< x >, < y >)$$

where $< x >$ and $< y >$ are multi-valued partial state records (PSRs), computed over one or more input data values, representing the intermediate state over those values that will be required to compute an aggregate. $< z >$ is the PSR resulting from the application of $M$ to $< x >$ and $< y >$. The merging function $M$ is required to be commutative and associative so that the aggregate is well-defined with respect to (unordered) relations. For example, if $M$ is the merging function for AVERAGE, each PSR will consist of a pair of values: SUM and COUNT, and $M$ is specified as follows, given
two PSRs \(< S_1, C_1 >\) and \(< S_2, C_2 >\):
\[
M(< S_1, C_1 >, < S_2, C_2 >) = < S_1 + S_2, C_1 + C_2 >
\]

The initializer \(I\) is needed to specify how to instantiate a PSR for a single input data value; for an AVERAGE over a data value of \(x\), the initializer \(I(x)\) returns the tuple \(< x, 1 >\). Finally, the evaluator \(E\) takes a PSR and computes the actual value of the aggregate. For AVERAGE, the evaluator \(E(< S, C >)\) simply returns \(S/C\).

\[\text{[JMH: Should we include Verification as a 4th function? If so, we should weave it through (including the trapezoid figure).]}\]

In addition to this traditional trio of functions, we will introduce an optional fourth verification function \(V'\), which takes a PSR and returns true iff the PSR is verified to be accurate. We will see examples of verification functions throughout the paper, and will provide a more general discussion in Section 6.2. We observe here that the verification function may need to examine local state and/or communicate over the network to make its decision.

The actual process of in-network aggregation involves multiple entities playing different roles, passing data from one to another; this is illustrated in Figure 1. These roles, and the message passing between them, work as follows:

- The querier is the agent that receives the final, fully merged PSR. It runs the verification function \(V'\) to check the PSR, and if it verifies then it executes the evaluator function \(E\) to generate the final query result. For ad-hoc queries, the querier is also responsible for initially disseminating the query request through the network, as described above.
- Sensors are agents that produce the raw data values for aggregation, and invoke initializer functions \(I\) to generate PSRs. In our scenario above, the “sensor” is the HID agent, and the data are measurements or alerts issued by that agent. We require that all the sensors are registered with a public key infrastructure (PKI) so that we can authenticate the data they produce.
- Aggregators combine multiple data PSRs through the use of the merging function \(M\). Note that the PSRs handled by an aggregator may distill differing amounts of sensed data from each input; the rather symmetric graph of Figure 1 should not be taken to mean that all aggregation topologies will be that regular. Proposed topologies for aggregation include trees [6], depth-based DAGs [1, 8], random point-to-point gossip [2] and hybrids of trees and DAGs [7]. In general, we will not make any assumptions about these communication patterns in our work, though we will comment further on the special case of DAGs in Section 6.2.

In our motivating scenario, these roles are distinct and may not be shared by a single principal. We return to this exclusion later on, when we address practical alternatives to this scenario. [PM: Check if we have and treat accordingly.]

2.3 Threat model

We turn now to the threat model that we address in this work, by identifying the tampering opportunities in the dataflow, as illustrated in Figure 1. First, the sensor can suppress data, or insert spurious data into the system. Second, an aggregator can take various actions to perturb the processing; these include suppression of PSRs, manipulation of PSRs (i.e., incorrectly executing the merging function \(M\)), and introduction of spurious PSRs. Finally, the finalization at the querier may be performed inappropriately, yielding results inconsistent with the input PSR.

In this paper we focus on defending vulnerabilities introduced by the aggregators, corresponding to the gray trapezoid in Figure 1. Before discussing those issues, we briefly discuss the other two vulnerabilities. At the bottom of the figure (vulnerability 1), the actual generation of raw input data by sensors is hard to protect in an application-independent manner. Although heuristics to verify input data may exist in particular environments (e.g., if it is known that accurate temperature readings in some sensor network deployment will never exceed 100F degrees), ultimately this task can only be performed by hardware, firmware, or software attestation of the data generators (see, for instance, work done in the Pioneer project [12]). For the purposes of this work, we focus on ensuring that the aggregates are faithfully computed over whatever raw data the sensors produce in the initializer, which is consistent with our motivating scenario.

With regards to finalization (vulnerability 3), the querier can still yield the wrong result even if aggregators are correct, by misapplying the evaluator function. Again, since our scenario assumes that the querier is the entity interested in the result, we do not examine

Figure 1: Dataflow among agents in the system, with potential threats by each agent listed in the black ovals. Proof sketches are used to thwart the threats within the gray trape- zoid. [JMH: Need to add verification function if we’re gonna use it. There is no such thing as a “Finalizer” any more. We should show each node running one of \(I, M, E\). We removed the term “PSR manipulation” from the threat model]
this vulnerability further. If required, the operation of the evaluator can be spot-checked, as is done in the context of SIA [9].

With regards to aggregation (vulnerability 2), the primary target of our work, there are two possible attacks. Partial-state suppression attacks are made by aggregators that omit data from input PSRs during the merging function. The simplest suppression attack is to suppress an entire PSR, but we will see examples where a complex PSR object (e.g. a set of sampled tuples) may have subparts that are suppressed. Spurious partial state attacks introduce into output PSRs data that should not be there; such data might be inappropriately merged input bits (e.g. a non-maximal value for a MAX aggregate [JMH: Is that what you had in mind?]), or data entirely invented by a misbehaving aggregator, such as an exemplar (e.g. for a MAX aggregate) that no sensor produced. [JMH: At this stage, it’s easy to see intuitions about exemplars, and hard to understand summaries. Which makes sense given the sequel, but should we acknowledge that or worry about it here?]

The current threat model cleanly encapsulates all threats within the aggregation infrastructure, by assuming sensors and the querier machine are trusted and can be authenticated. This is a reasonable assumption for our motivating scenario. It may have to be relaxed to generalize our results to a broader set of application settings, especially ones in which aggregators may collude with sensors. However, even in cases where this assumption does not hold, we note that our techniques still represent a significant contribution: they limit the attack vectors available to adversaries, and should work naturally in concert with solutions to other attack vectors, e.g. the Pioneer [12] approach for verifying input data. [JMH: Question for Petros: Is there some security apple-pie wisdom we can quote about the merits of research that plugs one hole in the world even if it doesn’t plug all of them?]

3. AM-FM PROOF SKETCHES

To begin our discussion of proof sketches, we consider the special case of the predicate polls discussed previously, in which we count the number of nodes that satisfy some boolean predicate. First, consider the standard trio of functions for the COUNT aggregate:

\[ I(x) = \langle x \rangle ; \quad \mathcal{M}(\langle x \rangle, \langle y \rangle) = \langle x + y \rangle ; \quad \mathcal{E}(\langle x \rangle) = x \]

A rogue aggregator can perturb the count in one of two ways: by inflating the value of a PSR during execution, or by deflating it. We treat these two cases separately; our protection for the inflationary case will serve as the basis for preventing deflation attacks as well.

3.1 Detecting Inflation via Authentication

To start, we need aggregators to prove that they did not inflate the result of the merging function. As one simple solution to the problem, we can change the aggregation logic to count in a unary representation. We modify the PSR for COUNT to be a bitmap of size \( U \), with one bit for each of the \( U \) sensors being polled. The aggregation logic is modified accordingly:

\[ I(x) = \langle 2^x \rangle ; \quad \mathcal{M}(\langle p \rangle, \langle q \rangle) = \langle p \lor q \rangle ; \]

\[ \mathcal{E}(\langle p \rangle) = |p| \]

where \( |p| \) is the count of 1-bits in the bitmap \( p \). To prove that the result was not inflated, we require each bit position to be cryptographically signed by its corresponding sensor. We augment the PSR with a set of signatures, one per bit; we refer to the set of signatures associated with a bitmap as its authentication manifest (AM). The initializer \( i \) initializes the AM to contain the signature for bit \( x \), and the merging function unites the AMs in addition to ORing the bitmaps. Given a PSR consisting of an AM and a bitmap, the querier can use the sensors’ public keys to verify that all the 1-bits in the bitmap are authentically signed, and hence that the count is not too high.

The obvious problem with this technique is that the size of the AM is on the order of the size of the number of nodes being counted, and the performance benefits of in-network aggregation are lost. This leaves us with a more focused challenge: how can we form a compact AM for COUNT?

3.1.1 AM-FM: Approximate Inflation Detection

To achieve this, we relax our security requirement: instead of requiring that we detect all inflationary attacks on the count, we can settle for detecting “noticeable” attacks that overcount by more than some small factor. This relaxation suggests the use of space-efficient Flajolet-Martin (FM) sketches for approximately counting the distinct values in a set [3]. By augmenting FM with an authentication manifest, we develop our first proof sketch, which we call AM-FM.

Quick Introduction to FM Sketches. The FM distinct-count estimator [3] is a one-pass (streaming) algorithm that relies on a family of hash functions \( \mathcal{H} \) for mapping incoming data values from an input domain \( [U] = \{0, \ldots, U - 1\} \) (e.g., node IDs) uniformly and independently over the collection of binary strings in \( [U] \). [JMH: What is a binary string in \( [U] ? \?) (The algorithm does not need to know \( U \) exactly — an upper bound on the domain size is sufficient.) It is not difficult to see that, for \( h \in \mathcal{H} \) and \( \mathsf{1} \mathsf{b}h(s) \) denotes the position of the least-significant 1 bit in the binary string \( s \), then for any \( i \in [M] \) \[ JMH: What is \( [M] ? \) \], \( \mathsf{1} \mathsf{b}h(i) \in \{0, \ldots, \log U - 1\} \) and \( \Pr[\mathsf{1} \mathsf{b}h(i) = l] = \frac{1}{M2^l} \). The basic FM-sketch synopsis (for a fixed choice of hash function \( h \in \mathcal{H} \) is simply a bit vector of size \( \Theta(\log U) \). This bit-vector is initialized to all zeros and, for each incoming value \( i \) in the input, the bit located at position \( \mathsf{1} \mathsf{b}h(i) \) is turned on. Of course, to boost accuracy and confidence, the FM algorithm employs averaging over several independent instances (i.e., \( r \) independent choices of the mapping hash-function \( h \in \mathcal{H} \) and corresponding FM sketches). A high-level description of the FM estimator is depicted in Figure 2.

Intuitively, due to the randomizing properties of the hash functions in \( \mathcal{H} \), we expect a fraction of \( \frac{1}{\log C} \) of the \( C \leq U \) distinct values in the stream to map to location \( l \) in each sketch; thus, we expect \( C/2 \) values to map to bit 0, \( C/4 \) to map to bit 1, and so on. Therefore, the location of the leftmost zero in a bit-vector synopsis is a good indicator of \( \log C \). In fact, Flajolet and Martin proved that the estimation procedure depicted in Figure 2 is guaranteed to return an unbiased estimate for \( C \) (i.e., the expected value of the returned quantity \( \hat{C} = E[\hat{C}] = C \).

The analysis of Flajolet and Martin assumes ideal randomizing hash functions

\[^2\text{All log’s in this paper denote base-2 logarithms.}\]
procedure FMDistinctEstimator (S, \{h_1,...,h_j\}) 
Input: Stream S of values in the domain [U] = \{0,...,U - 1\},
family of randomizing hash functions h_i (i = 1,...,r).
Output: Estimate \( \hat{C} \) of the number of distinct values in S.

begin
1. for \( i := 1 \) to \( r \) do
2. \( \text{fnSketch}(\text{ch}_i) = [0,0,0] \) // bitvector of size \( \Theta(\log U) \)
3. for each \( j \in S \) do
4. for \( i := 1 \) to \( r \) do \( \text{fnSketch}(\text{ch}_i)[\text{ab}(h_i(j))] := 1 \)
5. for \( i := 1 \) to \( r \) do
6. for \( m := \log U - 1 \) down to \( 0 \) do
7. if \( \text{fnSketch}(\text{ch}_m)[0] = 0 \) then \( \text{leftmostZero} := m \)
8. \( \text{sum} := \text{sum} + \text{leftmostZero} \)
9. endfor
10. \( \hat{C} := 1.2928 \times 2^{\text{sum}/r} \)
11. return (\( \hat{C} \))
end

Figure 2: The Flajolet-Martin (FM) Distinct-Count Estimator.

\[
I(x,a) = |< h(x) >| = |< h(a), x, a, s_a(x) >| \geq 2^{/h(h(x))} \tag{1}
\]

\[
M(p, A_p, q, A_q) = |< p OR q, A_p \cup A_q >| = |p \cup q, A_q > \tag{2}
\]

\[
\epsilon(p, A_p) = 2^{/p}/0.77351 \tag{3}
\]

Figure 3: Definition of AM-FM sketches as an aggregate. \( h \) is a hash function. \( b \) maps a bit vector to the bit position of its lowest-order 1-bit (e.g., \( b(0110100) = 2 \)). \( f \) maps a bit vector to the bit position of its lowest-order 0-bit (e.g., \( f(0110100) = 0 \)). \( A_p \) is the authentication manifest for FM bitvector \( p \). If \( p \)'s \( i \)-th bit is set to 1, the \( i \)-th component of \( A_p \) has the form \( < x, a, s_a(x) > \), and is correct if \( i = h(x) \) holds and \( s_a(x) \) is a valid signature on \( x \) by data source \( a \). The \( \cup \) operator forms the union of its inputs, and retains from the union only "exemplar" signature for each distinct bit \( b(h(x)) \).

7,? has shown that simple variants of the FM-sketch-based estimator can rely on much simpler, limited-independence hash functions (specified through concise, logarithmic-size random seeds). Using only \( y = O(\log(1/\delta)/\epsilon^2) \) basic FM sketches, these techniques give (randomized) \((\epsilon,\delta)\)-estimators for the number of distinct values \( C \); that is, the computed estimate \( \hat{C} \) satisfies \( \Pr[|\hat{C} - C| \leq \epsilon C] \geq 1 - \delta \) (?),(?).

Adding Verifiability: AM-FM Proof Sketches. From the perspective of verifiability, an attractive aspect of FM sketches is that each bit's value is an independent manifest of the input domain. Hence each 1-bit can be authenticated by a single signed value from the input that turns it on. So we can construct an authentication manifest for FM sketches with \( \log U \) or fewer signed inputs of the form \( < x, a, s_a(x) > \), one per 1-bit in the sketch, proving that sensor \( a \) provided value \( x \). We refer to the result as an AM-FM proof sketch; the details are given in Figure 3.

Discuss verification of the count not being too high.

3.2 Complementary Deflation Detection

The authentication manifest in AM-FM sketches prevents inflating the count by turning 0-bits in the FM sketch into 1-bits. The remaining possible attack is to turn 1-bits into 0-bits, and remove the corresponding signatures from the AM. This attack could deflate the count in the FM sketch.

One approach to preventing this attack is to use redundant communication to ensure that the signatures and 1-bits get through to the querier. Unfortunately, maintaining a sufficient degree of redundancy is expensive, as we discuss in Section 9.

Instead of relying on redundant communication, we rely on the querier having some information: the number of sensors \( U \). Given that information, we can accompany each predicate poll \( \text{pred} \) with its complementary poll \( \neg \text{pred} \), and simply check that their counts sum up correctly: \( \hat{C}_{\text{pred}} + \hat{C}_{\neg \text{pred}} = U \). We can accompany these counts with AM-FM proof sketches \( \hat{C}_{\text{pred}} \) and \( \hat{C}_{\neg \text{pred}} \) respectively. If the adversary deflects \( \hat{C}_{\text{pred}} \), he must inflate \( \hat{C}_{\neg \text{pred}} \) to avoid being detected by the sum check; however, the authentication manifest for \( \hat{C}_{\neg \text{pred}} \) prevents undetected inflation.

3.3 Verification and Analysis

The AM-FM proof sketch allows deterministic detection of spurious additions to the count. This is achieved by the querier first verifying the authenticity of each element of the authentication manifest by applying the appropriate private key, and then ensuring that the resulting decoded values can be used to reconstruct the FM sketch.

Given that the AM and the sketch match up, the remaining question arises from the use of FM approximations: how much "wiggly room" attacks does the inaccuracy in these approximations give an adversary interested in deflating the count?

To answer this specifically, assume FM-based estimators \( \hat{C}_{\text{pred}}/\hat{C}_{\neg \text{pred}} \) that use \( O(\log(2/\delta)/\epsilon^2) \) independent AM-FM sketch instantiations to estimate the Yes/No population counts (e.g., as discussed in [?]). Our aggregate verification step checks the condition \( \hat{C}_{\text{pred}} + \hat{C}_{\neg \text{pred}} \geq (1-\epsilon)U \) and flags an adversarial omission attack if the condition is violated. The following theorem and analysis establish the (probabilistic) error guarantees provided by our verifiable AM-FM aggregation scheme.

**Theorem 1.** Using \( O(\log(2/\delta)/\epsilon^2) \) AM-FM sketches to estimate \( \hat{C}_{\text{pred}} \) (and \( \hat{C}_{\neg \text{pred}} \)), and assuming a successful final verification step, the \( \hat{C}_{\text{pred}} \) estimate is guaranteed to lie in the range \( [\hat{C}_{\text{pred}} - \varepsilon U + \hat{C}_{\neg \text{pred}}] \), \( \hat{C}_{\text{pred}} > (1 + \varepsilon) \), \( \hat{C}_{\text{pred}} \leq \hat{C}_{\text{pred}} + 2\Delta U \), with probability \( 1 - \delta \). For predicate selectivities \( \geq \alpha \), this implies an \( \hat{C}_{\text{pred}} \) estimator for \( \hat{C}_{\text{pred}} \).

**Proof:** Consider the final estimation step for \( \hat{C}_{\text{pred}} \) at the finalizer. Note that our AM-FM sketches are naturally composable and duplicate insensitive summaries. Thus, assuming no malicious tampering with the FM portion of our sketch vectors, the summaries arriving at the finalizer are \( O(\log(2/\delta)/\epsilon^2) \) independent FM sketches over the underlying data population. Thus, earlier FM-based estimators (see, e.g., [?]) can be used to give \( \hat{C}_{\text{pred}} \), \( \hat{C}_{\text{pred}} \), and \( \hat{C}_{\text{pred}} \) for \( \hat{C}_{\text{pred}} \) and \( \hat{C}_{\text{pred}} \) (respectively); then, a simple application of the union bound implies that \( \hat{C}_{\text{pred}} + \hat{C}_{\neg \text{pred}} \leq (1 + \varepsilon)U \) with probability \( 1 - \delta \).

Now, let \( \emptyset \geq 0 \) denote the total (additive) underestimation error in the final \( \hat{C}_{\text{pred}} \) estimate, including both FM estimation error and omission error introduced by the adversary (through, possibly, several aggregator nodes) during the aggregation process. Thus, our
verification step estimates $U$ as $\hat{C}_{\text{pred}} + \hat{C}_{\text{pred}} = C_{\text{pred}} - \theta + \hat{C}_{\text{pred}}$. Then, if $0 > \varepsilon(U + C_{\text{pred}})$, with probability $\geq 1 - \delta$,

$$\hat{C}_{\text{pred}} + \hat{C}_{\text{pred}} < C_{\text{pred}} - \varepsilon(U + C_{\text{pred}}) + \hat{C}_{\text{pred}} \leq C_{\text{pred}} - \varepsilon(U + C_{\text{pred}}) + (1 + \varepsilon)\hat{C}_{\text{pred}} = (1 - \varepsilon)U$$

which implies that our sanity check will detect the adversarial omission attack (with high probability). The stated (worst-case) additive bounds for the $\hat{C}_{\text{pred}}$ estimate follow immediately. For the selectivity-based relative error bound, simply note that, if $C_{\text{pred}} \geq o(U)$, then $0 \leq \varepsilon(\frac{1}{\alpha} - 1)C_{\text{pred}}$.  

In other words, with our verifiable AM-FM aggregation scheme, any adversarial aggregator omission attack can cause our final $\hat{C}_{\text{pred}}$ estimate to underestimate the true count by at most $\varepsilon(U + C_{\text{pred}}) \leq 2\varepsilon U$, or risk being detected with high probability. Thus, the error guarantees for verifiable AM-FM estimate are in terms of $\varepsilon U$ factors, which are typically sufficient for predicates that represent significant fractions of $U$ (e.g., predicates for quantile ranges or “heavy-hitters” [17, 7]). Specifically, for predicates with selectivity $\geq \alpha$, our algorithms can give $\varepsilon(\frac{1}{\alpha} - 1)$-relative error bounds with probability $\geq 1 - \delta$. It is important to note here that the authentication-manifest portion in our sketches play a crucial role in our error bounds, by essentially forcing the adversary to inject only one-sided (omission) error — otherwise, the adversary could arbitrarily inflate $\hat{C}_{\text{pred}}$ while lowering $\hat{C}_{\text{pred}}$ (or, vice versa) thus making arbitrarily large errors essentially undetectable.

3.4 Leveraging COUNT

Here we explain any stuff we can do in the one-value-per-node setting based on counting, including:

- Fixed-width Histograms (i.e. group by)
- [MHH:What Else? Isn’t heavy-hitters/iceberg more like distinct sample? Frequent items too, right?]

4. VERIFIABLE RANDOM SAMPLING: THE AM-SAMPLE PROOF SKETCH

We turn to the slightly more involved problem of constructing a verifiable random sample of a given size $k$ over the data tuples residing at the leaf nodes (sensors) in our distributed aggregation topology (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1). Note that such a sample represents a general-purpose summary of the sensor contents that can be employed at the querier (Figure 1).

A conventional random-sampling summary is simply a pair $(\{t_1, \ldots, t_k\}, N)$ comprising (a) the subset of sampled tuples, and (b) the total count of the underlying population (sampling rate $= k/N$). Such a sample can be dynamically generated moving up our aggregation architecture, e.g., using a simple adaptation of reservoir random sampling [2]. Unfortunately, it is not difficult to see that such a scheme falls short of our verifiability goals. Specifically, consider an aggregator node in our architecture (Figure 1) receiving two random samples $(s_1, N_1)$ and $(s_2, N_2)$ from its children. Even though it may be possible to authenticate individual tuples in $s_1$ and $s_2$ (e.g., through hash signatures), an adversarial aggregator can still introduce arbitrary bias in the sample by simply “cheating” on the reservoir-sampling algorithm. Consider, for instance, the above scenario with $|s_1| = |s_2| = k$ (the target sample size) and $N_1 >> N_2$, and a malicious aggregator that deterministically outputs $(s_2, N_1 + N_2)$ rather than sub-sampling $s_1$ and $s_2$ with the appropriate rates; furthermore, even if $N_1 = N_2$, the adversary could bias the sample towards specific data values (e.g., choose the $k$ smallest sensor readings in $s_1 \cup s_2$) resulting in arbitrarily-biased approximate query answers.

The key problem with such conventional sampling schemes in our setting is that they essentially offer no means to verify the sampling procedure run at each aggregator (i.e., the validity of each aggregator’s random coin flips). Instead, our proposed AM-Sample proof sketches collect a random sample by employing hash functions to map (data tuple, sensor-id) elements to buckets with exponentially-decreasing probabilities, as in FM estimation. The key difference with simple AM-FM sketches is that we now retain authentication manifests for all exemplar elements mapping above a certain bucket level $l$ (along with their respective level) — these are exactly the elements in our sample. Our sampling scheme is similar in spirit to Gibbons’ distinct-sampling technique [7] for approximating COUNT DISTINCT queries over data warehouses; essentially, by forcing elements to be distinct (through the addition of the sensor-id field), we can use similar ideas to collect a verifiable random sample over the underlying data tuples.

Formally, given a uniformly randomizing hash function $h$ over (tuple, sensor-id) pairs and a target sample size $k$, an AM-Sample proof sketch comprises a pair $< L, S >$, where

$$S = \{(t_1, < t_1, a_1, s_{a_1}(t_1) >), \ldots , (t_m, < t_m, a_m, s_{a_m}(t_m) >)\}$$

is a subset of $m \leq 2k$ tuple authentication manifests $< t_i, a_i, s_{a_i}(t_i) >$ with corresponding bucket levels $l_i = \lfloor b(h(t_i, a_i)) \rfloor$. The semantics of an AM-Sample sketch (assuming no tampering) is that it stores exactly the authentication manifests (and bucket levels) for (tuple, sensor-id) elements at levels greater than or equal to $L$ (which, of course, implies the invariant $l_i \geq L$ for all $i = 1, \ldots, m$). Since each element maps to a bucket level $l$ with probability $1/2^{l+1}$, it is not difficult to see that each element in the AM-Sample sketch is chosen/sampled with probability $\sum_{L \leq l} 1/2^l = 1/2^L$.

A concise TAG-like description of our in-network aggregation scheme for AM-Sample proof sketches is given in Figure 4. In a nutshell, given a target sample size of $k$, our algorithm starts by computing the authentication manifests (and bucket levels) for individual sensors (Equation (4)). These manifests are then unioned up the aggregation tree, by appropriately sub-sampling elements at higher sampling rates (using the maximum level $\max(L_1, L_2)$ to build the output sample); furthermore, to keep the sketch size under control, our aggregation scheme drops the sampling rate by a factor of 2 (setting $L = \max(L_1, L_2) + 1$) when the sample size grows beyond $2k$ (Equation (5)).

The gist of the technique is to collect all distinct values that would turn on each FM bit position in AM-FM (Section 3), instead of exemplars for each bit position. After an aggregator fuses all of its input PSRs, if the resulting PSR contains more than $2k$ values, then the aggregator must drop some values from the current sample. Starting with the rightmost 1-bit in the sketch, the aggregator “prunes” that bit, dropping all values in the current sample that would turn that bit on. The process is repeated with higher-order FM bits, until the current sample size has dropped below $2k$. Figure 4 describes this process precisely.

Verifiable distinct sampling combats crimes of commission in two
Figure 4: Definition of AM-Sample proof sketches as an in-network aggregate.

ways. First, it prevents the adversary from inventing new values, since all values come signed with the signing key of an authorized data source. Second, it prevents the adversary from migrating a value to an unpruned value set, since inclusion of a value in a value set is contingent on the signed content of the value, and cannot be forged after its source has signed it.

Verifiable distinct sampling combats crimes of omission in two ways. First, because it partitions the universal value set in sub-sets of exponentially decreasing cardinalities (in expectation), it places a lower bound on the cardinality of each value set: the i-th value set is expected to contain $\frac{U}{2^i}$ values, and the delivered sample with a pruning level of $l$ is expected to contain $U(1 - \sum_{i=0}^{l-1} 1/2^{i+1}) = U/2^l$ values. Second, huh?

### 4.1 Worst-case Analysis

### 5. VERIFIABLE CASCADED SKETCHES

### 5.1 Design

### 5.2 Worst-case Analysis

### 6. GENERALIZING

### 6.1 Beyond Polling

[JMH: Here we talk about allowing each node to have an arbitrary number of tuples. Should we do this earlier?]

### 6.2 A Generalized Template for Proof Sketches

To develop a proof sketch for an aggregation function $\gamma$, the following steps are required:

1. **Omission Protection:** An authentication manifest is to protect against commission attacks. A key challenge in designing a proof sketch is to develop a compact manifest with few witnesses, and still provide bounds on the possible aggregation error an adversary could introduce via the discarded witnesses. [JMH: Can philosophize about how it’s good to have each bit being verified be dependent on a small “support” in the input. AMS sketches don’t work nicely because of this.]

2. **Omission Protection:** For some aggregate $\gamma$, our complementary omission protection requires:

   (a) the querier can verifiably maintain $\gamma(U)$

   (b) derive an analytic bound on the effect of omitted data, given the value $\gamma(U) - \gamma(P) - \gamma(P)$, you can reliably detect omitted data.

Note that it is possible to use an approximate caching scheme to maintain $\gamma(U)$ (e.g., [? ]). The bounds provided by approximate caching must be incorporated into the bounds provided for the possible error due to omission attacks.

[JMH: This is a good place to talk about duplicate insensitivity and the possibility of doing topological omission protection, at a significant cost.]

### 7. EXPERIMENTAL EVALUATION

### 7.1 Threat Models

### 7.2 Results

### 7.3 Discussion

### 8. EXTENSIONS

### 8.1 Accountability

If I know the subtree sizes.

On-the-fly accountability.

Detection in the second pass.

### 8.2 Early Commitment

Early commitment.

### 8.3 Efficient Signature Computations

Bundle signatures of all values into incremental structures to avoid individual value signing.

### 9. RELATED WORK

### 9.1 Multipath Omission Protection (MOP)

The proof sketches we develop here are duplicate-insensitive aggregates [1, 6, 8].

like MIN and MAX may be copied and redundantly routed through multiple paths in a network, without requiring any bookkeeping to remove duplicate messages in order to correctly evaluate the final result. A variety of techniques have exploited this property to mitigate the effects of communication failure during distributed aggregate computation. Our vUNION aggregate is duplicate-insensitive: its merging function chooses the same exemplar for a value $x$ regardless of the number of times each exemplar is merged in. Hence multipath routing is one possible strategy for preventing omission attacks on vUNION. We refer to this approach as Multipath Omission Protection (MOP).

While the direction is encouraging, multipath routing has not been explored in an adversarial setting for distributed aggregation, and provides no guarantees on avoiding determined adversaries. [JMH: Double-check that assertion!] How do we guarantee that the network topology is robust against adversaries? In the presence of up to $k$ adversarial nodes, omissions can be prevented by routing each message along $k + 1$ node-disjoint paths to the finalizer. This requires maintaining $k + 1$-connectivity between each data generator and the finalizer. This is a special case of finding the min cut in a graph, which can be done using the distributed algorithm given by Goldberg and Tarjan [4]. However, this is not inexpensive in practice, running in $O(n^2 \log n)$ time for a network of $n$ nodes. Moreover, practical distributed protocols to maintain $k$-connectivity in the face of node arrival and departure are not very scalable; typical results
are for $k = 2$ or $k = 3$ [?]. [JMH: Need better citation here. See survey by Eppstein/Galil/Italiano.]

All of the proof sketches we present in this paper are duplicate-insensitive, and are amenable to MOP. However, since the state of the art with respect to adversary-proof multipath routing has limitations, we now explore an additional protection against omission, which works under a different set of conditions.

9.2 other stuff

Differentiate from: privacy preserving stuff, trustworthy storage, trustworthy base data, including watermarking and “web of trust” kinds of beliefs about data. It should be possible to accommodate such technologies in our scheme (e.g., weighting the contributions of participants?)

Sanli et al. [11] describe machinery for secure key distribution and aggregation-aware encryption in sensor networks. They set encryption parameters so as to provide greater security (higher-quality encryption) the closer an aggregator lies to the root of the aggregation topology and, therefore, the greater fraction of the aggregated sensed values that aggregator summarizes. Unlike AM-FM, this work does not consider the problem of aggregators acting maliciously.

10. FUTURE WORK AND CONCLUSIONS

Deal with topology, control path, loss.

Acknowledgments

Remember to acknowledge Aydan Yumerefendi, David Wagner, Dahlia Malkhi.

11. REFERENCES


