Appendix 1

A.1 Privacy Enhancing Technologies
In this section, the authors will discuss several privacy enhancing technologies that were part of the TIPPERS system deployed in the US Navy ship. In particular, the section explains the differential privacy, secure database, and policy technologies along with a description of their usage in the context of Trident Warrior 2019.

A.1.1 Differential Privacy
PeGaSus (PGS) is a specific algorithm for analyzing streaming data under differential privacy [17]. PGS supports a variety of analyses; counts, sliding windows, and event monitoring, in real time with provable guarantees of privacy and low error. It also supports analysis at multiple resolutions, from an individual sensor to aggregate statistics of a collection of sensors.

PGS processes a set of data streams in short windows (say 5 minutes). At the conclusion of each time window, data is aggregated into a set of counts. For example, one count might report the number of individuals observed by a particular sensor at least once during the previous 5 minutes. These counts are fed into the *Perturber*, the first of three components in the PGS system. The Perturber adds random noise to the counts, where the noise is sufficient to ensure the differential privacy guarantee. Practically speaking, this means that the location of a user at a particular time is effectively hidden. While the noise ensures privacy, it may also create artificial effects in the data stream, such as jitter, even when the underlying data stream is smooth.

The two remaining components of PGS address the issue of spurious patterns arising from the noise infusion. The *Grouper* partitions the data into smooth windows -- intervals of the data stream where the total deviation from the window average is small. Crucially, this grouping must be done carefully to ensure the DP condition is met. The final module is the *Smoother*, which combines the outputs of the Perturber and Grouper, to produce a data stream where spurious jitter from the noise has been greatly reduced.

The Perturb-Group-Smooth technique, the inspiration for the name PeGaSus, is data-adaptive; it yields higher accuracy on streams that are smooth, but can nevertheless detect changes in the pattern (such as a conference room transitioning from unoccupied to occupied).

A.1.2 Secure Databases and Encrypted Storage
There are two specialized databases that are incorporated into TIPPERS: PULSAR and Jana. In this section, the authors will discuss those technologies and how they function.

A.1.2.1 PULSAR Technology
A standard secret-sharing scheme allows a dealer to randomly split a secret into two or more shares, such that certain subsets of the shares can be used to reconstruct the secret and others reveal nothing about it. The simplest type of secret sharing is 2-party additive secret sharing, where the secret s is an element of an Abelian group G, and it is split into two shares (r,r-s) that add up to s, where r is a random element of G. A useful feature of such scheme is that it is homomorphic in the sense that if many secrets are shared, the two parties can individually compute shares of the sum of the secrets by locally adding their shares, without any communication. This feature of additive secret sharing (more generally, linear secret sharing) is useful for many cryptographic applications.

The notion of function secret sharing (FSS) [10,11,18] can be viewed as a natural generalization of additive secret sharing to functions, where the challenge is to keep the shares succinct. Concretely, suppose a class F of functions f mapping n-bit strings to elements of G is provided. Is it possible to split an arbitrary f from F into two functions, f1 and f2, such that: (1) f=f1+f2 (on every input x), (2) each of the two shares has a short key describing it that enables its efficient evaluation, and yet (3) each key alone hides f? A scheme as above is referred to as an FSS scheme for the class F. A special case of interest is the class of point functions f, which have a nonzero output on at most one input. An FSS for this class is called a distributed point function (DPF).

If one insists on perfectly hiding f, then it can be shown that, even in the simple case of DPF, the best possible solution is to additively share the truth-table representation of f, whose shares consist of 2^n group elements. But if one considers the computational notion of hiding, then there are exponentially better solutions in many cases of interest, including the DPF case.

Efficient constructions of FSS for simple classes can be constructed from minimal cryptographic assumptions such as a block cipher (e.g. AES). This can be applied to private reading (e.g. finding a nearby restaurant without revealing your location [19]), private storage (e.g. anonymous messaging [20]). One exemplary application is that of secure distributed histograms or "distograms", that allow for the ability to privately aggregate information into histogram buckets. Stealth (makers of PULSAR) incorporates these distograms into the PULSAR (Private Updateable Lightweight Scalable Active Repository) solution as a means of real-time privacy-preserving data aggregation and retrieval which can be applied to sensors, mobile devices, WiFi access points, and beyond.

New research has shown that FSS can also be used for computing more complex functions such as in the work of Boyle, Gilboa, and Ishai [11] as well as important applications in secure RAM computation such as Doerner and Shelat [20] and Bunn, Katz, Kushilevitz, and Ostrovsky [21].

A.1.2.2 Jana Technology

Jana technology implements the paradigm of Private Data as a Service (PDaaS). Jana technology provides order revealing encryption in the form of POPE indexes. This allows Jana to use its database query engine to efficiently process operators such as range selection, at some quantifiable cost in information leakage. (Note that humans are particularly poor at understanding impacts of information leakage. Thus, even when such leakage is quantifiable it remains an open research question to determine whether the information is at all useful to data contributors or systems security analysts). Jana also supports encrypted secret shares suitable for linear secret sharing based processing of query operators using the SCALE/MAMBA MPC engine, allowing the technology to process data with no leakage at some quantifiable cost in query performance. Uniquely, Jana uses linear secret sharing methods to compute and apply its differential privacy protections, so that query results are never “in the clear " within the system.

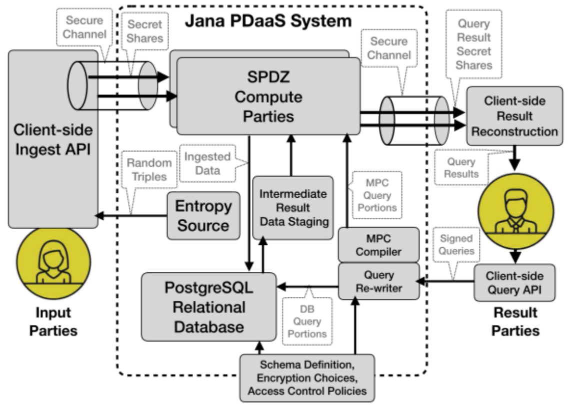

Figure 15. Jana PDaaS System.

Jana is illustrated in Figure 15. The bottom center is the commercial relational database, PostgreSQL, where encrypted (or plaintext, if desired) data is stored in relational form.
One or more contributing parties, shown at left in the figure, use their platforms to encrypt data (with help from Jana) before actually sending the data on to Jana. At top right, it is shown that Jana receives queries. Using the list of available encryptions stored in the database for each attribute in each relation, our query re-writer modified each query, choosing whether to execute each query operator using the relational database query engine, or the SCALE/MAMBA MPC engine. If deterministic or order revealing encryptions are usable by a query operator, and are available for the attributes accessed by the operator, then the re-writer runs applicable operators in the Postgres database directly (and more efficiently than in MPC). For other operators, our re-writer retrieves necessary secret shared data from the relational store, and then issues byte-code programs that evaluate those operators over the retrieved data in SCALE. Note that application of differential privacy randomness is always performed in the SCALE engine. In each case, the final result of a query is produced in secret shared form for communication to the querier. These result shares are then combined by client-side Jana tools to provide query results in usable form.

Jana also provides and enforces a role-based access policy to control who can ask queries and receive query results. Jana includes a language for describing what outcomes should be made available, with what protections applied, to queries based on attributes assigned to them by the DBA. Available protections are specifiable at the per-record and per-attribute level of granularity. Protections include preclusion of access (no access), allowance for access (full access), aggregate-only access, and differentially private aggregate-only access.

A.1.3 Policies and Policy Enforcement

In TIPPERS, policies are used to guide the collection, storage, processing, and sharing of data. These policies are either defined by the administrator of the space and therefore apply to any device in it (e.g., due to security reasons) or defined by users to express how their data should be managed (e.g., to restrict access to pieces of information about them). In this demonstration the primary focus is on the former types of policies as in this specific environment there is a strong requirement to enforce the policies of the space such as preventing unauthorized users from entering a certain area or detect when a fire code violation is taking place. This way, a sample policy in the system is defined to alert the administrator using the command control application of situations where there is an unauthorized access to a space.

This policy is defined through the TIPPERS interface and enforced by leveraging the Pulsar system. To this end, a query is continuously posed to Pulsar to return true if the count of people with a specific badge (which represents whether the authorization level of a person) in specific spaces (e.g., the armory) is different from zero. Pulsar computes this count operation efficiently and returns an answer to the app which, in case of a violation, displays an alert.

With respect to user-defined policies, the system denies access to an individual's data by default when an application tries to access it. As an example of user-defined policies, the team created a policy to enable the group commander to access data about people in his/her group. This policy, would be defined by each member of the group, grants explicit permission to the group commander to access the location of the person. Internally, these policies are managed by the TIPPERS policy engine that enforces them at query time. In particular, as discussed in Section 2.1, TIPPERS offers the possibility to define such policies in high-level terms (e.g., restricting access to location data instead of to specific sensor data) and translates it into access control to low-level sensor data. This way, if a user tries to access, for instance, connectivity records of a specific device, this will be denied if the user defined a policy to restrict access to his/her location as this low-level data can be used to infer such information [7].

A.1.4. Details of PETs and TIPPERS usage in Trident Warrior exercise

As explained in Section 2.1, the TIPPERS system integrates the previously explained PETS and decides how to use them depending on the specific piece of data collected. In summary, the PeGaSus technology from UMass/Colgate/Duke is used to generate differential private streams of data into TIPPERS. In the context of the Trident Warrior exercise, a PeGaSus virtual sensor is used in TIPPERS to generate differential private occupancy counts over synthetic data generated for the ship. Notice the ability to only instrument with real sensors in a few spaces and thus, synthetic presence and occupancy data was generated for the rest of spaces (i.e., other rooms/floors). As Figure 2 shows (see Section 2.2), sensor data flowing from the synthetic data generator tool, as well as from the real sensors, is translated into presence through our *Connectivity2Presence* virtual sensor. Then, this data is further processed to generate occupancy counts by another virtual sensor (*Presence2Occupancy*) at different space granularity levels i.e. region, floor, room. All these streams of occupancy data are passed to the PeGaSus virtual sensor to generate a differentially private version. There exist two instantiations of the PeGaSus virtual sensor with two different epsilon values (0.5 and 1.0) that generate a stream of differentially private occupancy with different noise levels.

A web application was developed to visualize the differentially private occupancy data and compare it with the real occupancy data in order to study its utility [23]. This tool, is based on a TIPPERS application (Building Analytics) that was built to give access to a building administrator to better plan spaces and events, as well as to better control HVAC systems in order to be more energy efficient without having access to the real data. In the tool, the user has the option to view differentially private occupancy data for different time intervals and space granularity inside the building.

It was decided to use the secure databases developed by Stealth and Galois (makers of Pulsar and Jana, respectively) for two specific tasks. The first task stored real counts of occupancy (divided by badge profile -- i.e., visitor, worker, etc.) and performed policy checking on the encrypted domain. The second task stored metadata, such as who is the host of each visitor as well as their current location in the encrypted domain. Figure 2 (see Section 2.2), shows how the data flows into both systems, as well as the data models defined for them. Notice that a new virtual sensor was developed to compute occupancy data of each space divided by profile. This information is pushed into the Pulsar system. Also, the real-time presence data generated by the virtual sensor using connectivity data is pushed into the Jana system, along with metadata about the relationship between hosts and visitors.