Addressing Data-Centric Security Requirements for IoT-Based Systems

Juan D. Parra Rodriguez*, Daniel Schreckling†, Joachim Posegga‡
Institute of IT-Security and Security Law
University of Passau, Innstraße 43, Passau, Germany
Email: *dp@sec.uni-passau.de, †ds@sec.uni-passau.de, ‡jp@sec.uni-passau.de

***Abstract*—Allowing users to control access to their data is paramount for the success of the Internet of Things; therefore, it is imperative to ensure it, even when data has left the users' control, e.g. shared with cloud infrastructure. Consequently, we propose several state of the art mechanisms from the security and privacy research fields to cope with this requirement.**

To illustrate how each mechanism can be applied, we derive a data-centric architecture providing access control and privacy guaranties for the users of IoT-based applications. Moreover, we discuss the limitations and challenges related to applying the selected mechanisms to ensure access control remotely. Also, we validate our architecture by showing how it empowers users to control access to their health data in a quantified self use case.

***Index Terms*—Internet of Things; Security Architecture; Data-Centric Security; Differential Privacy; Secure Cloud Storage; Encryption**

I. Introduction

In response to the highly unbalanced resource distribution in the Internet of Things (IoT), resource-constrained devices tend to only collect data while cloud servers store and process high amounts of information. Though practical for developers and for data availability purposes, this raises security and privacy issues. Particularly, because data is delivered to the cloud in plain text, thus making it impossible for users to perform access control on their data.

In spite of extensive research to address security challenges for the IoT [1], [2], [3], a unified data-centric solution to ensure that users can perform access control on their data, even outside of their computers and devices is currently not available. Notwithstanding, there is extensive research on several fields of security and privacy such as homomorphic encryption, trusted computation, or differential privacy that could help IoT-based application to provide some access control and privacy guarantees for the user. As a result, we aim to start the journey towards a security framework addressing data-centric security requirements on access control and privacy by proposing an architecture including such mechanisms.

Our contributions are summarized as follows:

- we present a data-centric security architecture, for systems using IoT gateways, that empowers users to control access to their data. The security architecture has been defined considering several attacker models. To synthesize the attacker models we based the architecture on perimeters including components trusted by the user to handle his data.
- our security architecture applies current state of the art privacy and security mechanisms to its different components. More to the point, these mechanisms are the result of a survey to find techniques applicable for the IoT set-ups covered by our architecture.
- we present a concrete validation scenario for our proposal through a quantified self use case. The main goal is to allow users to control who has access to their data through technical means, yet permitting external systems to perform aggregation operations from information provided by users opting in. More to the point, information disclosed by users for aggregation purposes contemplates the use of mechanisms to protect the user's identity and privacy.

The rest of this paper is organized as follows. We present relevant security requirements and the architecture for IoT-based applications used for our analysis in Sections II and III. Afterwards in Section IV, we present the attacker models based on different perimeters including trusted entities; moreover, this section includes a discussion on perimeter enforcement based on secure hardware and software combinations. Then, Section V lists state of the art software-based mechanisms helping to tackle security requirements in particular perimeters. Later on, we integrate every software-based mechanism into a unified security architecture and show how it can be applied to a quantified self use case in Section VI. Last but not least, we cover the limitations of our architecture and our conclusions in Sections VII and VIII.

II. Security requirements

Herein we describe the set of data-centric security requirements addressed by our security architecture:

R1 A user can grant access to data to other users while keeping it confidential for unauthorized entities.

R2 The identity associated with the data being accessed can only be determined by a set of authorized entities.

R3 A user can partially expose data to external entities in order to protect his privacy.

Notably, the biggest challenge is to ensure these security requirements in spite of data being stored and processed by external systems, and in spite of the distributed nature of IoT-based applications.

 DOI: https://doi.org/10.1109/SIoT.2016.007

III. Architecture

This section defines terminology for the architectural components that will be used throughout the paper (See Figure 1).

IoT Device is considered to be an electronic platform, e.g. Arduino or a sensor, with some wired or wireless connectivity such as WiFi, IEEE 802.15.4, or an analogue or digital wired connection.

Gateway is an electronic system, such as an ODROID or a Raspberry Pi, running a full-fledged operating system and communicating not only with IoT devices, but also with the Internet. The purpose of the gateway is twofold: it enables devices with different protocols to interact with each other, and at the same time it can process data for aggregation, analysis or security purposes and preform temporary data storage. Since gateways need to keep physical proximity with IoT devices and sensors, they are likely to be deployed inside a NAT (Network Addressing Translation) network, e.g. at home. As a result, they do not necessarily offer services to the internet through inbound connections due to NAT and firewall restrictions.

External System can be any application on the Internet offering services to the gateway through an API using protocols such as CoAP, HTTP, etc. We call such systems *external* as they are not in direct control of the user but are managed by a third-party. External systems include but are not limited to email, social networks and weather forecast services. Moreover, these systems can include some storage capabilities to fulfill the application's goal. However, even though some storage-related calls could be invoked by the gateway, e.g. create a post on a social network, the external system would do some kind of processing on the data, therefore controlling data in terms of format, size, etc. Please note that an external system can also consist of another gateway and multiple IoT devices behind it.

External Storage is a term we use to refer to systems offering data storage functionality. Such systems can offer enhanced CRUD (Create Read Update Delete) APIs, or any kind of storage and querying interface, such as SQL. Examples of such systems include Dropbox, Google Drive, or a MySQL database, etc. In contrast to external systems which manipulate input data following a specific business logic, external storage system store and return data as provided by the user.

Visualization Device refers to any kind of platform allowing users to visualize and process data coming from IoT Devices, Gateways, External Systems or Storage. These platforms are commonly Web browsers, mobile phones, tablets, etc. Although a direct connection between a visualization device and a gateway is possible (assuming that network connectivity from the visualization device to the gateway is available), we have decided to leave it out of the architecture to consider a realistic scenario coping with the network restrictions described in the gateway's definition. Further, it is assumed that the IoT device, the gateway, the external system and the visualization device can be programmed in a Turing complete language.

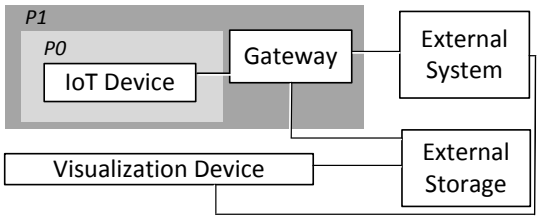

Fig. 1. IoT System Architecture and Security Perimeters (P0,P1,P2,P3)

IV. Perimeters and Attacker Models

Given that users may desire more or fewer security guarantees, depending on what they consider an acceptable trade-off between functionality and security, we use the concept of a *Perimeter* as the set of components in the architecture trusted by the user. First, this section will define the perimeters considered in this paper based on a general system architecture we consider for IoT systems. Second, we will justify the choice of our perimeters by outlining the numerous options on how such perimeters can be supported by soft- and hardware technologies which already exist or are under development.

A. Perimeters and Attackers

As described above, Figure 1 outlines the general IoT system architecture we consider. It also depicts the four perimeters we consider in this contribution. They reflect components trusted by the user. In our paper, this implies that a user relies on the fact that an attacker is unable to perform sophisticated physical hardware attacks on a device and that the execution of any code on the device does not break data secrecy. Trusted components inside a perimeter are also assumed to securely transmit data to properly authenticated entities inside the perimeter. This renders the consideration of active or passive insider attackers trying to manipulate or eavesdrop communication unnecessary.

Hence, our perimeter model proves useful to clearly define where the attacker can be: Outside of the perimeter. Further, we can indicate the power of an attacker by indexing perimeters. Perimeters with a given index i are always a proper subset of perimeters with j when $i < j$. For instance, in P0 it is assumed that an attacker cannot interfere with the IoT device; conversely, in P0 an attacker could tamper with the gateway, the external system, external storage and the visualization device.

Additionally, our perimeter model also indicates the trust level a user is prepared to assign to specific components. A user which only accepts a perimeter of level 0 is more *paranoid* than a *laid-back* user who trusts every possible component participating in the processing of his data and who is comfortable with a maximum perimeter of level 3.

Below, we sketch various technologies which show that such perimeters can actually be implemented and indicate in which perimeter they can be used to secure it.

B. Perimeter Enforcement

Processing of accumulated data is fundamental for IoT-based systems. As the applications in the IoT domain are manifold and so are the program logics behind it. All components in our high-level architecture, apart from the external storage component, can execute some use case specific and possibly untrusted, third-party code. Further, if authorized, data may be transmitted to systems running malware or to systems whose operating system has been compromised. Hence, apart from a mere trust in specific manufacturers or software, the decision whether a component can be considered trusted is hard and would require considerable effort. We consider trusted execution environments (TEE) as a feasible solution for this purpose.

In the best case a TEE should ensure the following properties [4]. The integrity and secrecy of software modules executed by a TEE and the data it processes should be protected. TEEs should also provide storage in which confidential data of a module can be stored securely while the module is not executed. Through remote attestation, clients which are required to exchange messages with specific software modules should be able to verify that they are in fact talking to this particular module. To be able to actually talk to such modules, a TEE should ensure secure provisioning of software modules, i.e. entities are able to send data to specific modules on specific devices while maintaining their secrecy and integrity. Finally, a TEE should enable the authentic and confidential communication between software modules and device peripherals.

The size and complexity of the trusted computing base (TCB) required for each architectural component will depend on the number and type of properties implemented in it. We will list some of the most prevalent technologies we consider feasible to be deployed in IoT environments and indicate in which perimeter they could be used for enforcement.

1) TEE with Micro-Controller Support: A TEE can be implemented by means of complex hardware and software support. However, constraints on resources and price often render such solutions infeasible. Thus, approaches such as SMART [5], TrustLite [6] or TyTan [7] exploit properties of low-end micro controller units or of recent CPU technologies feasible for IoT devices to build simplified TEEs. SMART introduces a new primitive based on low-end micro controller hardware and software TCB which protects tasks through read-only memory and provides attestation mechanisms. To achieve this, it uses ROM as a secure storage with minor hardware modifications and some adjustments to the memory management in the controller itself. Of course, this renders the secure storage to be vulnerable against physical attacks. Further, the approach does not allow changes to tasks and relies on the micro controller to sanitize memory when access violations occur. TrustLite also aims for embedded devices but gives greater flexibility. It extends SMART by supporting task interruption and dynamic changes on the attested code. However, TrustLite dynamics is also limited as it requires the configuration, loading, and isolation of all software components at boot time. TyTan [7] removes the later restriction of TrustLite, allows for the dynamic loading of modules and is able to give real-time guarantees.

Trusted execution environments specifically designed for embedded devices are particularly feasible for IoT devices in perimeter P0. While a user may simply trust such simple devices a TEE can ensure the correctness of firmware updates, protect data against possibly malicious software accidentally installed by the user or by the exploitation of a vulnerability in the gateway.

2) TEE with Trusted Hardware Support: McCune et al. introduce Flicker [8] which exploits the dynamic root of trust measurement (DRTM) enabled by trusted platform modules (TPM) [9] or mobile trusted modules (MTM) [10], TPMs for mobile devices. A static root of trust measurement (SRTM) ensures system integrity beginning with the boot process, i.e. it measures all software loaded since the loading of the BIOS. The idea is to subsequently rely on the OS to perform process isolation. This implies, that the whole OS becomes the TCB which appeared to be infeasible. DRTM was designed to solve this problem by being able to dynamically reset the CPU into a fixed state and start measuring from the reset on. This concept was intended to also support a hypervisor based system where the hypervisor would be measured and attested by the TPM. Unfortunately, hypervisors are still complex and the TCB has considerable size.

Thus, Flicker tries to further reduce the TCB by introducing pieces of application logic (PAL) which must be defined and implemented by the developer of an application. Whenever these PALs must be executed, Flicker suspends the OS, executes the PAL, allows the manipulation of secure storage, increments the instruction pointer stored in the TPM, seals all security critical storage again and then restores the OS. As Flicker runs with highest privileges it can protect the PAL from the OS, even if the latter was compromised.

Flicker introduces large overhead due to its intense use of the TPM. Thus, McCuner further refines his DRTM based model by introducing a hypervisor called TrustVisor [11] which uses software-based μTPMs which are associated with PALs. In contrast to a hardware TPM which runs in an often slower and completely isolated piece of hardware, the μTPM runs on the main CPU. As the TrustVisor is managing PAL and μTPM execution it is measured by the physical TPM which builds a new root of trust, the TrustVisor root of trust measurement (TRTM). While Flicker only protects the PAL from the OS also enables the protection of the OS from the PAL. However, the non-security sensitive part of the application, i.e. code not inside a PAL, may still contain malware that can compromise OS.

We believe that the approaches introduced above can benefit from newer CPU based technologies, such as the ARM TrustZone[12], Trustsonic[1] or Samsung KNOX[2], and the Intel

[1]https://www.trustonic.com/

[2]https://www.samsungknox.com/

Software Guard Extensions (SGX) [13]. The functionality and computing power of a TPM is restricted and the physical binding with the devices is rather weak. Therefore, Flicker and TrustVisor could further benefit from the power and security features of these platforms. With SEDA [14], Asokan et al. already show how the Intel research architecture Siskiyou Peak for embedded devices can support powerful and efficient security mechanisms such as swarm attestation for low-level devices by enhancing SMART and TrustLite.

In this way, trusted execution environments supported by trusted hardware become a valuable enforcement mechanism for P0-P3.

3) TEE with Trusted Hardware and Virtualization Support: Of course, various virtualization techniques can be deployed in powerful devices [15]. Thus, perimeter P2 and P3 can easily deploy virtualization mechanisms which enable the isolation of possibly malicious code and control the access to security critical data and credentials.

However, virtualization is also a valid option for mobile and embedded devices our architecture puts in P1 and P2. This has been shown by numerous implementations such as TrustDroid [16] and Boxify [17]. Also the solution by Wessel et al. [18] which improves mobile device security by deploying operating system-level virtualization becomes an attractive option for these perimeters as it provides storage encryption, integrity protection and remote management.

Combined with a TPM or more advanced processor technologies the measurement of appropriate hypervisors or snadboxes mentioned above could form a dynamic root of trust which can be asserted by a connecting client.

In fact, the more recent solution Sprobes by Ge et al. [19] exploits the ARM TrustZone and introduces introspection without a separate hypervisor. While this approach only shows how to limit kernel execution to approved code pages in order to prevent rootkit exploits, this work introduces a lightweight solution by exploiting modern processor architectures. Although this approach requires dynamic code rewriting, we assume that this work can be further exploited and applied to approaches implementing TEEs by controlling memory access with program counter.

Another recent approach combines TrustVisor (see above) with the Google Native Client (NaCl) [20]. NaCl uses a combination of a secure runtime and software-based fault isolation (SFI) [21] to sandbox native application code. The combination with TrustVisor and thus the use of a TPM generates a two-way sandbox called MiniBox [22]. It allows the attacker to control applications as well as the operating system. Thus, MiniBox can provide sufficient trust to be placed in P1-P3. Through its TPM and μTPM usage it can also provide attestation to a client within the same perimeter.

Finally, we want to highlight an approach which uses fully abstract compilation. Fully abstract compilers translate fully indistinguishable source level programs into indistinguishable target level programs. This property is also called contextual equivalence and assumes programs at source level to be correct, i.e. if they do not allow data leaks, the target level program will possess the same property. While this assumption requires additional implementation effort from the developer fully abstract compilation provides fine granular access control on data and ensures data secrecy and integrity. Implementations for the fully abstract compilation of object-oriented programming languages into untyped assembly target languages exist [23], [24].

Particularly intersting is the work by Strackx et al. Similar to TrustVisor, they define self-protecting modules (SPMs) [25]. They can be executed using shared resources such as memory or CPUs, can authenticate towards other modules, can securely communicate with other modules and ensure that the sensitive information they store can only be modified by code from the module itself. While their first contribution requires special hardware support their second approach, Fidel [26], targets commodity systems. To achieve this, they complement their work with an additional hypervisor, a small kernel which enforces fine-grained data access and a fully abstract compiler able to generate secure SPMs. Similar to TrustVisor a TPM can provide the required DTRM during the launch of the hypervisor. Both models allow powerful attackers which can control the operating system and can inject malicious software or other potentially hostile SPMs. However, as for almost all approaches, attackers which can run physical hardware attacks are excluded in their attacker model.

Of course, both approaches are feasible for perimeters P1 through P3. On top, depending on the device configuration these approaches could also be deployed in P0. In particular, Fidel requires only limited hardware support and provides tremendous security guarantees for data secrecy at the same time.

V. Existing Software-Based Mechanisms

This section will describe state of the art mechanisms addressing security requirements described in Section II. After describing each mechanism, we state in which components of the architecture it can be applied, the perimeter according to the IoT System Architecture in Figure 1, and the security requirements addressed by it. Further, when a mechanism can be applied in several perimeters, we describe the mechanism assuming the strongest attacker model. i.e. the smallest perimeter possible, in which the mechanism could be used.

A. Cryptographic Mechanisms

1) Homomorphic Encryption: The basic principle behind homomorphic encryption is to use a transformation, i.e. encryption function, that preserves structure, i.e. an homomorphism. That is to say, given an encryption function f, such that $f(x+y) = f(x) \oplus f(y)$, an external party which does not know x and y can perform the addition of two encrypted values $f(x)$ and $f(y)$, by using the operator $\oplus$. Since $f(x) \oplus f(y)$ produces $f(x+y)$, the actual value for the sum $(x+y)$ can only be decrypted by users with the proper key to reverse f.

One of the biggest challenges of homomorphic encryption is to support polynomials of arbitrary degree efficiently. As a result, it is common to see homomorphic encryption tailored

to particular applications in which feasible performance is achieved [27], [28], [29].

Given that the gateway and the visualization device are likely to have less computational resources than external systems, we focused our survey on server-side processing on encrypted data. The key contribution of such approaches is that gateways and visualization devices can offload storage and processing of data to external systems, yet maintaining their data encrypted at all times. The most relevant approaches for server-side aided homomorphic encryption are HElib [30] and CryptoNets [31]. The former is a library supporting SIMD (Simple Instructions Multiple Data) operations homomorphically and specifying their cost. This library has been produced in C and C++ and it has been open sourced under GPL Licence. CryptoNets allows to apply neural networks to encrypted data using homomorphic encryption, though it is not available as open source.

Encrypting and decrypting data on the gateway and the visualization device while performing homomorphic operations on the external system could be applied to perimeter P2. Moreover, homomorphic encryption tackles R1 while letting the gateway and the visualization device to offload computation and storage of data to an external system.

2) Searchable Encryption: Song et. al proposed a set of cryptographic schemes for searching on encrypted data [32]. Moreover, they demonstrate that an attacker who reads the cipher text cannot learn anything about the plain text. Further they provide controlled searching, i.e. the server cannot perform a query without the user's authorization, and also the user can ask the server to search for a secret keyword without revealing it to the server. Also, algorithms presented by Song et. al are practically applicable, given that they have $O(n)$ number of stream cipher and block cipher operations for a message of length n.

The theoretical foundations presented by Song et. al [32] were leveraged by CryptDB [33], along with additional schemes such as order-preserving and homomorphic encryption, to implement SQL-aware cryptographic schemes. This allows CryptDB to support SQL queries on encrypted data; however, developers must specify beforehand the primitives used by the application's queries, e.g. join, and the particular fields they are used in. Unfortunately, this is required to guarantee effective query resolution; for instance, if one requires a SORT BY aggregation for a particular column, order-preserving encryption on the column is naturally required to resolve the query. According to the authors' evaluation, CryptDB supports 99.5% of the queries observed in a 126 million SQL trace extracted from production systems. Also, CryptDB reduces the throughput by 14.5%, thus making it an acceptable solution for server-side encrypted storage. CryptDB also supports multiple principals by chaining keys, e.g. encrypted the group key with each user's key, so they can decrypt the group's key while keeping it confidential to third parties.

CryptDB's architecture relies on two components: a server side database which performs queries on encrypted data, and a proxy used by the application to generate and send encrypted queries, generated from standard SQL statements, to the server. Therefore, data confidentiality is protected when the database server is compromised, since the server can neither decrypt the query, nor the result. On the other hand, when the proxy is compromised, the confidentiality of data belonging to users who are currently logged cannot be ensured; notwithstanding, data readable by users who are not currently logged in, through the compromised proxy, is kept confidential.

Another approach to encrypt data on the user's side while allowing the server to perform searches on the encrypted data is Mylar [34]. Mylar is implemented as a plug-in for the Meteor JavaScript framework, which uses elliptic curves to encrypt data on the user's browser while allowing users to send encrypted keyword searches to the server. Moreover, in the process of resolving the keyword search, the server neither learns the query, nor the plain text of the result. Like CryptDB, Mylar also supports data sharing through chaining if keys, as CryptDB does. On top of this, this framework also considers integrity protection for parts of the code hosted by untrusted servers; to this end, a browser plug-in verifies signatures of the sources loaded from the application while ensuring proper isolation of the cryptographic keys from untrusted servers through the Same Origin Policy.

The applicability of all concepts described in this subsection lies in perimeter P2 and they would address requirement R1. However, there are subtle differences depending on which method is applied. If CryptDB is applied, the database would comprise an external storage component, and the proxy used for the SQL translation would be executed on the gateway and the visualization device. On the other hand, if Mylar is applied, the developer needs to be able to modify the behaviour of the gateway, the visualization device and the external system. The later is required because Meteor not only generates the browser-side code, i.e. visualization device, but also generates the code for the external system to host the application and the data, along with specific business logic required the search on the encrypted data set.

B. Partial Data Exposure

1) Differential Privacy: According to the literature [35], [36], differential privacy operates on databases containing rows where the data of an individual is held in a single row. Further, differential privacy ensures that the ability of an adversary to inflict harm on any set of people is essentially the same, independent of whether any individual opts in or out of the dataset. This is done by focusing on the probability of any given output of a privacy mechanism and how this probability can change with the addition or deletion of any row.

Differential privacy mechanisms rely on adding some kind of noise to the data, in order to prevent the disclosure of the exact value provided by a participant on the dataset, while producing acceptable results when data is aggregated. This is possible because noise is canceled out during the aggregation process for particular probability distributions. However, most of the work on differential privacy assumes a database in

which a participant answers a particular question only once (provides only one value). Intuitively, this is a prerequisite to ensure that data cannot be "averaged" to find the actual value for a participant over time, i.e. longitudinal attacks. This poses a critical pitfall for the IoT domain because IoT devices will generate data periodically, and it is not feasible to assume that each device provides a single measurement in its lifetime. Therefore, only very particular mechanisms such RAPPOR [37] and the work on differential privacy under continual observation by Dwork et. al [36] can provide guaranties against untrusted aggregations attempting a longitudinal attack on measurements received over time. These two approaches can be tuned by modifying parameters on the client-side to find the proper trade-off between usability and privacy.

On the one hand, RAPPOR provides support for collecting one or more categorical responses, i.e. whether a particular entity belongs to a category or not. RAPPOR could also be used to collect population statics on numeric and ordinal values, i.e. use predicates associated with particular range of values. Further, RAPPOR can be used to calculate statistics on non-categorical domains or when the categories are not known in advance through Bloom filters. On the other hand, differential privacy under continual observation [35] allows to count how many times a certain event has occurred in the past.

Both of the aforementioned differential privacy techniques could be applied to perimeter P0 by including them in the IoT device itself to support R3. Hence, it must be noted that for both schemas the systems doing the aggregation do not need to be in the perimeter because even if the decoding of the data is not implemented properly, e.g. Bloom filter for RAPPOR, there are no threats against confidentiality.

2) Sanitization: Sanitization is a term commonly used when sensitive data is removed (or redacted) from a document to reduce privacy risks. Recently, the National Institute of Standards and Technology (NIST) released a technical report on de-identification techniques [38]. In our architecture, sensitive data could be removed at any point. The smallest perimeter that could be used is P0. Applying de-identification or sanitization solves requirement R3 and requirement R2. Further, under the right conditions, de-identification and sanitization could provide a stronger property than R2 because the identity associated with the data cannot be determined by anyone; however, whether this possible in practice is still an ongoing debate due to the difficulty to ensure that data remains unlinked to other datasets that could ultimately reveal the identity of the data owner.

C. Hidden Channels

Lulia et. al propose a Web-based protocol for users who want to communicate with each other through existing external systems, e.g. Facebook, while keeping information confidential and hiding from a casual observer that such confidential exchange of data takes place [39]. In a nutshell, in the scheme proposed by Lulia et. al a user can publish a "regular" message, and encode information in the message in such a way that the recipient can fetch an encrypted file. This file can be hosted in a publicly available Dropbox or Google Drive folder and it is referenced by a URL shortener service such as tinyurl. Once the user, with whom data should be shared with, has downloaded the encrypted file, it is subsequently decrypted with the user's key. The whole process of publishing, retrieving, and replacing the original message in the graphical interface (HTML) is transparently performed by a browser plug-in. An interesting fact of this approach is that other users who are not aware of how data from the "regular" message should be decoded will not suspect that an encrypted message is being shared between the creator of the message and another user. Furthermore, even if a user who is not intended to read the data manages to find the location of the encrypted file, he still will not be able to read the clear text from it.

For clarity, let us assume a scenario in which Alice manages the gateway and she wants to send an encrypted message to Bob holding a visualization device. In this scenario, the gateway would perform the encryption. Afterwards, the gateway would encode the location of the encrypted file in a "regular" message and post it to the external system, e.g. Facebook. Subsequently, the gateway would use an external storage system, e.g. Dropbox, to make the file available to Bob. Once Bob reads the "regular" messages from the external system, his visualization device should then decode location from the public external storage, fetch the file and decrypt it with Bob's key. The last step could be performed through browser plug-ins or applications running on the visualization device. Moreover, communication in the other direction would work symmetrically. Lulia et. al have shown how this mechanism can be implemented sharing messages through Facebook and other online services.

The method described previously can be applied to perimeter P2; however, the functionality available as a browser plug-in needs to be implemented in the gateway and the visualization device. Furthermore, this approach addresses requirement R1, and the key differentiator of this technique is that R1 is achieved without modifying the external system.

D. Anonymity

1) Onion Routing: Tor (The Onion Router) is an open-source, circuit-based and low-latency service for anonymous communication [40]. Roughly speaking, Tor allows a sender to choose a multi-hop circuit comprised of Onion Routers to send his message. Messages are encrypted using several layers to ensure that Onion Routers participating in the circuit can forward the message, yet without having access to the data being exchanged. Furthermore, since the sender of the message chooses the path it is very hard for nodes to collude and brake the anonymity of the sender. In Tor's architecture there is a key component taking care of circuit establishment and message exchange installed on the sender's machine called the Onion Proxy.

Onion Routing could be applied to perimeter P2. On the one hand, the visualization device could keep its anonymity, thanks to the plethora of possibilities to install Tor clients for Visualization devices [41], [42], [43]. On the other hand, a

gateway could also be configured to send network requests through Tor anonymously. This is possible by configuring the gateway to send every network request through the SOCKS5 [3] interface of the Onion Proxy; alternatively, specific libraries can be used to overwrite network-related system calls, so they can be redirected through Tor, for any application [44]. Onion Routing addresses requirement R2 in a stronger sense because no one is be able to find the sender's identity. However, care must be taken to avoid sending information that may uniquely identify the sender of the data, and also a proper number of nodes in the network should be ensured.

2) Garlic Routing: The Invisible Internet Project (I2P) is an open-source anonymous network allowing applications to send messages to each other [45]. Unlike Tor where Onion Routers are not necessarily run by regular users, I2P requires that every user accessing the I2P network runs his own "router" to join the network. Also, applications accessed through this network need to use the proper I2P interfaces to communicate with the outside world. Another key difference between Tor and I2P, is that I2P uses Garlic Routing which uses a Distributed Hash Table (DHT) instead of centralized directory servers. As a result, when a router joins the network, it establishes a set of inbound and outbound tunnels used to relay messages, and each user can decide how many hops are required to find an acceptable trade-off between anonymity and usability.

Similarly to Onion Routing, Garlic Routing could be applied to perimeter P2 since I2P offers libraries for mobile devices. Furthermore, I2P can be executed in the gateway since it is implemented in Java.

3) Pseudonyms: There is a recent IETF draft for privacy-enhanced tokens proposing a schema for "Pseudonym-based Authorization Tokens" [46] based on one-way functions, e.g. cryptographic hash. In this protocol, tokens do not reveal any information about the client. Furthermore, by observing messages exchanged, including the tokens, an observer cannot know whether two messages were generated by the same sender. Most importantly, with the pseudonym-based authorization tokens, only an entity with the proper key material can find out the real identity of the sender, and an attacker who obtains a token cannot derive valid tokens from it.

This approach could be applied to perimeter P0, since in some IoT devices it is currently possible to evaluate cryptographic hash functions, and this would improve the privacy of an IoT device towards the gateway. Alternatively, the same argument applies between the gateway and an external system, or even between any pair of devices communicating in the architecture, e.g. visualization device communicating with external system. Moreover, in cases when this technique is applied but IP networks are used, anonymity of the client could only be preserved by applying also the Onion or Garlic Routing approach described in V-D1. Pseudonyms address requirement R2 since only a selected set of users can derive the identity of the token bearer, i.e. entities who posses the proper key material.

[3]SOCKS5 is a protocol to exchange network packets through a proxy server

VI. Security Architecture

We extend the initial architecture shown in Figure 1 to tackle the access control and privacy requirements from Section II. Subsequently, we apply our security architecture concepts to a quantified self use case and show its advantages.

A. Big Picture

In the security-enhanced architecture shown in Figure 2, the mechanisms introduced in Section V are placed in components where they are applicable. Besides, to illustrate against which attacker model they are effective, each mechanism is coloured following the same color coding used for the security perimeters. For instance, the partial exposure mechanism in the IoT device is coloured with the weakest gray, i.e. the same as perimeter P0, to show that applying it to the IoT device protects the user's data against attackers outside of the gateway. Furthermore, we use the color for the strongest attacker model, i.e. smallest perimeter, against which the mechanism is effective: in the example, we chose P0 even though partial data is also effective for to P1, P2, and P3.

In cases when different characteristics are required for client and server side, e.g. homomorphic encryption, we indicate which side is applied to each component. Also, in the case of hidden channels only clients are visible because this technique neither modifies the external system, nor the external storage.

In our architecture, both the external storage and external system include searchable encryption capabilities. This is required because the same goal can be achieved by an application hosting the code and the encrypted data, i.e. Mylar, or encrypted data could stored by an external storage and available through an SQL interface, i.e. CryptDB. Furthermore, since most of the mechanisms applied to the gateway are also required for the visualization device, in order to achieve secure communication, they are also drawn in each component.

B. Illustrating Use Case

The proposed data centric security architecture is applicable in the quantified self use case, where the main concept is tracking aspects of a users daily life (i.e food calories, health status, heart rates, oxygen levels etc) and his performance (i.e. physical activities, calories burnt, motion data etc). Such data may be collected by IoT devices, e.g. step tracking bracelets and smart watches, and sent to the IoT gateway (Raspberry Pi) using Bluetooth Low energy (BLE) or other wireless protocols. In the specific use case scenario, a company proposing a holistic approach to the quantified self concept provides an external system with an integrated Personal Health Record (PHR), which is accessible through a Web browser, and a native smart-phone application to interact with the user. Users of this system can instruct their gateways to upload data from their IoT devices in encrypted form to the external system. Furthermore, as specified in requirement R1, they can selectively share parts of their data with their caring circle including their physicians, relatives and friends, etc. Also, whenever the user wants to visualize his quantified self data (which is stored encrypted by server), he can use his

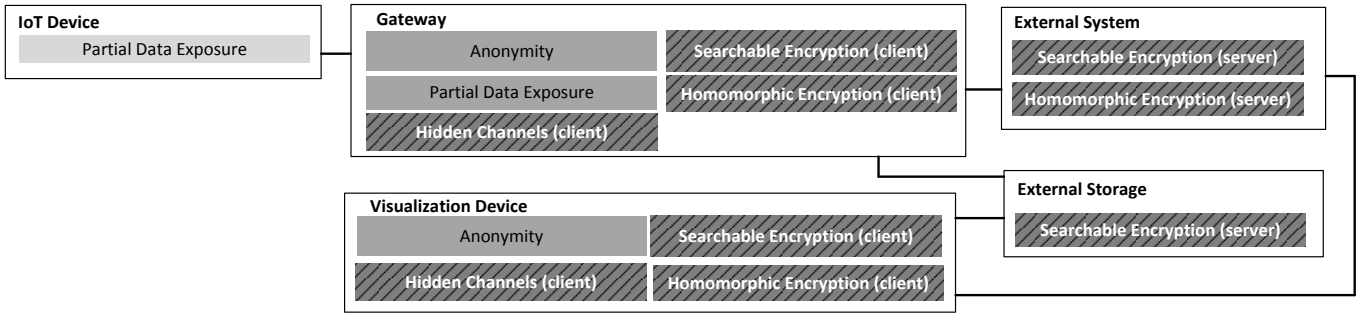

Fig. 2. Security Enhanced Architecture

visualization device (Web browser or smart-phone). This can be achieved exploiting searchable encryption on the external storage (or external system depending on whether Mylar or CryptDB is used) , and placing the proper functionality on the gateway and the visualization device as well, so as to decrypt the information and send encrypted queries to the server. In this particular use case, we are assuming that the user of the quantified self system feels comfortable with perimeter P2.

It is likely that the company developing the holistic quantified self application (external system + smart-phone application) would be motivated to provide additional services, e.g. at additional cost, based on aggregated values resulting from analysing the data from several devices and users. This is not possible once our proposal for searchable encryption has been implemented. Nonetheless, a middle ground can be implemented to allow the quantified self application to calculate aggregated values with acceptable accuracy while protecting the users' privacy, even against a subpoena from the government, i.e. requirement R3. Users could have incentives, e.g. lower monthly costs, to share their data de-identified and using differential privacy, which could be easily applied by the gateway under the user's control. In turn, this would allow the external system to earn revenue based on aggregated values using data only from users who have opted in to share their data while protecting the users privacy. An interesting aspect of our proposal is that hinders an external system from harvesting the users data without their knowledge through technical means. However, this also has a silver-lining for companies supporting such mechanisms: users will feel safer knowing that their data can only be used when they opt in, instead of by default therefore improving the application's popularity. At the same time, the company implementing the quantified self use case cannot be forced to deliver confidential data from its customers by any government because it is encrypted in the first place.

VII. Limitations and Challenges

Although mechanisms listed in this paper help to improve privacy and security guaranties for users, there are important considerations to keep in mind and tackle when possible. These considerations include ensuring proper system bootstrapping, password recovery among others.

Bootstrapping of a secure IoT-Based system remains an important challenge. Key agreement and out-of-band delivery of Trusted Execution Environments such that they can be remotely attested later on should be carefully implemented. Furthermore, mechanisms for replacing Trusted Execution Environments due to failures, among other reasons, and ensuring that this does not become the weakest link in the security chain needs to be assessed for each particular implementation.

Another critical point calling for a practical solution is that most of the mechanisms described in Section V rely on chaining keys to store group and users' keys. For instance, when the server stores a key encrypted with the user's password, the user's life is simplified because he can log in from any Web browser, while keeping the user's key confidential from the server. However, if the user forgets his password there are no means to recover this key. In the end, this problem boils down to an ancient problem balancing safety versus security. On the one hand, if one stores the key in plain text, the user can always log in (safety), but someone may have access to this key and then harm the user (security). So, in summary users of such secure systems should be well educated and keep secure backups from their passwords and keys, since strong guarantees for secrecy come with a price, i.e. there is no "I forgot my private key" mechanism. Further, additional issues such as interoperability, software stability and performance are key aspects for the integration.

Regarding software maturity, it is clear that Tor has been extensively used by real users showing its stability. At the same time, there are many systems currently using building blocks from CryptDB [47]. On the other hand, there are other components in the architecture which are either not open source, e.g. CryptoNets, or have not been yet thoroughly tested in real-life use cases yet, e.g. HELib.

From the performance point of view, some components mentioned in the architecture have been already used from hardware commonly used for IoT gateways; for example, Tor or I2P can be installed on a Raspberry Pi [48], [49]. Nonetheless, in spite of coming a long way since its initial definition on 1978 by Rivest [50], homomorphic encryption still faces an important performance challenge. Recently, framework for testing homomorphic encryption schemes named HETest was introduced [51]. In this work, it is stated that although

encryption and decryption with HELib is fast, i.e. order of tens of milliseconds, performing homomorphic evaluations can take hours.

Applicability of homomorphic encryption is hindered by its requirement to compute circuit-based operations, rarely seen in real-life. Also, applying cryptographic schemes such as CryptDB require care regarding the kind of data stored [52].

Last but not least, an unavoidable fact is the so called *analogue loophole* mentioned in Digital Rights Management (DRM). The analogue loophole is defined as the intrinsic possibility of copying and distributing content (or data) through analogue means once it is represented in a human readable format. For instance, regardless of the number of software and hardware-based mechanisms employed to share confidential information from an IoT device to a visualization device, the user receiving the data can always take a picture and post it online, therefore breaking the secrecy of the data completely. Although this is impossible to solve, it is relevant to clarify the limits of the technical mechanisms presented herein.

VIII. Conclusions

We have reached a point in which several security concepts that were only theoretically possible are becoming feasible in practice. A clear example is allowing users to query on their encrypted data without revealing the clear text to the server hosting it (See Section V).

A similar argument holds for the trusted execution environments feasible for the IoT domain. Expensive trusted platform modules with restricted functionalities start to vanish. They are replaced by cheaper and faster multi-purpose processing units. As sketched above, their combination with recent research results which introduce sophisticated compilation and virtualization technologies, is able to further reduce the TCB and increase the performance as well as security assurance even in the absence of trusted elements of platform modules. Although, many of these new devices still lack the ability to counter strong physical hardware attacks and thus may leak confidential data they become an attractive alternative to expensive and slow tamper-resistant hardware. We assume that this also holds for manufacturers of embedded devices. Instead of integrating trust anchors offered by only a small number of specialized suppliers, cheaper solutions with a reduced set of fabrication requirements but with distinct security benefits may be a clear advantage in the competition for a mass market. In particular, this holds for solutions which offer a physical integration such as the ARM TrustZone.

In general, compromises are required to reach applicability of security and privacy systems. For example, the cryptographic methods compiled in this paper can support confidentiality, yet they cannot guarantee integrity or availability of the encrypted data. This cannot be prevented because, in spite of not being able to decrypt the information stored in the external system, a malicious administrator could still modify it or even delete it.

All in all, we have focused on a specific set of requirements on access control and privacy. It has been particularly interesting to consider the IoT-based system as a whole, i.e. including external systems. After exploring state of the art mechanisms, we conclude that there are enough applicable mechanisms to improve the security guarantees for the users' data when external systems are willing to apply them.

Acknowledgements

This research has been supported by the EU under the H2020 AGILE (Adaptive Gateways for dIverse muLtiple Environments), grant agreement n. H2020-688088. Additionally it was partially funded by the "Bavarian State Ministry of Education, Science and the Arts" as part of the FORSEC research association. Further, the authors would like to thank Ilias Maglogiannis and Andreas Menychtas for providing valuable input for the use case validation.

References

[1] R. Roman, P. Najera, and J. Lopez, "Securing the Internet of Things," *Computer*, vol. 44, no. 9, pp. 51–58, Sept 2011.

[2] S. Sicari, A. Rizzardi, L. Grieco, and A. Coen-Porisini, "Security, privacy and trust in Internet of Things: The road ahead," *Computer Networks*, vol. 76, pp. 146 – 164, 2015.

[3] E. Vasilomanolakis, J. Daubert, M. Luthra, V. Gazis, A. Wiesmaier, and P. Kikiras, "On the Security and Privacy of Internet of Things Architectures and Systems," in *2015 International Workshop on Secure Internet of Things (SIoT)*, sep 2015, pp. 49–57.

[4] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. M. McCune, *Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 159–178. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_10

[5] K. Eldefrawy, A. Francillon, D. Perito, and G. Tsudik, "SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust," in *NDSS 2012, 19th Annual Network and Distributed System Security Symposium, February 5-8, San Diego, USA*, San Diego, UNITED STATES, 02 2012.

[6] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A Security Architecture for Tiny Embedded Devices," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: ACM, 2014, pp. 10:1–10:14.

[7] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny Trust Anchor for Tiny Devices," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 34:1–34:6.

[8] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for Tcb Minimization," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, ser. Eurosys '08. New York, NY, USA: ACM, 2008, pp. 315–328.

[9] Vincent J. Zimmer and Shiva R. Dasari and Sean P. Brogan, "Trusted Platforms - UEFI, PI and TCG-based firmware," White Paper by Intel Corporation and IBM Corporation, September 2009.

[10] Trusted Computing Group, "TCG Mobile Trusted Module Specification (Version 1.0, Revision 1)," https://www.trustedcomputinggroup.org/specs/mobilephone/tcg-mobile-trusted-module-1.0.pdf, June 2007.

[11] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "TrustVisor: Efficient TCB Reduction and Attestation," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 143–158.

[12] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using ARM Trustzone to Build a Trusted Language Runtime for Mobile Applications," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 67–80.

[13] Intel Corporation, "Intel® Software Guard Extensions (INTEL SGX)," https://software.intel.com/sites/default/files/332680-002.pdf, June 2015, accessed: 2016-06-10.

[14] N. Asokan, F. F. Brasser, A. Ibrahim, A. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "SEDA: scalable embedded device attestation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 964–975.

[15] E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan, "Security of os-level virtualization technologies: Technical report," *CoRR*, vol. abs/1407.4245, 2014. [Online]. Available: http://arxiv.org/abs/1407.4245

[16] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and Lightweight Domain Isolation on Android," in *1st ACM CCS Workshop on Security and Privacy in Mobile Devices (SPSM'11)*. ACM, 2011.

[17] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky, "Boxify: Full-fledged App Sandboxing for Stock Android," in *24th USENIX Security Symposium*. USENIX, 2015.

[18] S. Wessel, F. Stumpf, I. Herdt, and C. Eckert, "Improving Mobile Device Security with Operating System-level Virtualization," in *28th IFIP International Information Security and Privacy Conference (SEC 2013)*, 2013, accepted for publication.

[19] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture," *CoRR*, vol. abs/1410.7747, 2014.

[20] B. Yee, D. Sehr, G. Dardyk, B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar, "Native Client: A Sandbox for Portable, Untrusted x86 Native Code," in *IEEE Symposium on Security and Privacy*, IEEE, 3 Park Avenue, 17th Floor, New York, NY 10016, 2009.

[21] U. Erlingsson and F. B. Schneider, "SASI Enforcement of Security Policies: A Retrospective," in *Proceedings of the 1999 Workshop on New Security Paradigms*, ser. NSPW '99. New York, NY, USA: ACM, 2000, pp. 87–95.

[22] Y. Li, J. McCune, J. Newsome, A. Perrig, B. Baker, and W. Drewry, "Minibox: A two-way sandbox for x86 native code," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 409–420.

[23] P. Agten, R. Strackx, B. Jacobs, and F. Piessens, "Secure compilation to modern processors: extended version," Department of Computer Science, KU Leuven, CW Reports CW619, April 2012, partner: KUL; project: NESSoS; tier: NoTier; citations: 1.

[24] M. Patrignani, P. Agten, R. Strackx, B. Jacobs, D. Clarke, and F. Piessens, "Secure Compilation to Protected Module Architectures," *ACM Trans. Program. Lang. Syst.*, vol. 37, no. 2, pp. 6:1–6:50, Apr. 2015.

[25] R. Strackx, F. Piessens, and B. Preneel, "Efficient Isolation of Trusted Subsystems in Embedded Systems." in *SecureComm*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, S. Jajodia and J. Zhou, Eds., vol. 50. Springer, 2010, pp. 344–361.

[26] R. Strackx and F. Piessens, "Fides: Selectively hardening software application components against kernel-level or process-level malware," in *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*. ACM Press, October 2012, pp. 2–13. [Online]. Available: https://lirias.kuleuven.be/handle/123456789/354603

[27] P. Hallgren, M. Ochoa, and A. Sabelfeld, "InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol," *2015 13th Annual Conference on Privacy, Security and Trust, PST 2015*, pp. 1–6, 2015.

[28] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '10. New York, NY, USA: ACM, 2010, pp. 48–59.

[29] C. Wang, K. Ren, J. Wang, and K. M. R. Urs, "Harnessing the Cloud for Securely Solving Large-Scale Systems of Linear Equations," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, June 2011, pp. 549–558.

[30] S. Halevi and V. Shoup, "Algorithms in HElib," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8616 LNCS, no. PART 1, pp. 554–571, 2014.

[31] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," Microsoft Research, Tech. Rep. MSR-TR-2016-3, feb 2016. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=260989

[32] D. X. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, ser. SP '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 44–. [Online]. Available: http://dl.acm.org/citation.cfm?id=882494.884426

[33] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting Confidentiality with Encrypted Query Processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. New York, NY, USA: ACM, 2011, pp. 85–100.

[34] R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, and H. Balakrishnan, "Building Web Applications on Top of Encrypted Data Using Mylar," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 157–172. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/popa

[35] C. Dwork, M. Naor, T. Pitassi, G. N. Rothblum, and S. Yekhanin, "Pan-Private Streaming Algorithms," *Proceedings of The First Symposium on Innovations in Computer Science (ICS 2010)*, pp. 1–32, 2010.

[36] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," in *Proceedings of the Third Conference on Theory of Cryptography*, ser. TCC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 265–284.

[37] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response," in *Proceedings of the 21st ACM Conference on Computer and Communications Security*, Scottsdale, Arizona, 2014.

[38] S. L. Garfinkel, "De-Identification of Personal Information," NIST, Tech. Rep. NISTIR 8053, 2015. [Online]. Available: http://dx.doi.org/10.6028/NIST.IR.8053

[39] F. Beato, I. Ion, S. Čapkun, B. Preneel, and M. Langheinrich, "For Some Eyes Only: Protecting Online Information Sharing," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 1–12.

[40] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21.

[41] "Orbot Proxy with Tor," https://play.google.com/store/apps/details?id=org.torproject.android, accessed: 2016-06-10.

[42] "iTunes Onion Browser," https://itunes.apple.com/en/app/onion-browser/id519296448?mt=8, accessed: 2016-06-10.

[43] "Tor Browser," https://www.torproject.org/projects/torbrowser.html.en, accessed: 2016-06-10.

[44] "TorSocks," https://github.com/dgoulet/torsocks/, accessed: 2016-06-10.

[45] "I2P: The Invisible Internet Project," https://geti2p.net/en/, accessed: 2016-07-10.

[46] J. Cuellar, S. Suppan, and H. Poehls, "ietf-draft: Privacy-Enhanced Tokens for Authorization in ACE," https://www.ietf.org/id/draft-cuellar-ace-pat-priv-enhanced-authz-tokens-00.txt, June 2015.

[47] "CryptDB," https://css.csail.mit.edu/cryptdb/, accessed: 2016-06-10.

[48] "Onion Pi: Make a Raspberry Pi into a Anonymizing Tor Proxy," https://learn.adafruit.com/onion-pi/install-tor, accessed: 2016-08-10.

[49] "I2PBerry allows Raspberry Pi users to surf the I2P anonymously," https://www.element14.com/community/community/raspberry-pi/blog/2014/07/22/i2pberry-allows-raspberry-pi-users-to-surf-the-i2p-anonymously, accessed: 2016-08-10.

[50] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On Data Banks and Privacy Homomorphisms," *Foundations of secure computation*, pp. 169–177, 1978.

[51] M. Varia, S. Yakoubov, and Y. Yang, *HEtest: A Homomorphic Encryption Testing Framework*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 213–230. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-48051-9_16

[52] M. Naveed, S. Kamara, and C. V. Wright, "Inference Attacks on Property-Preserving Encrypted Databases," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: ACM, 2015, pp. 644–655. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813651