The Blind Enforcer: On Fine-Grained Access Control Enforcement on Untrusted Clouds*

Dinh Tien Tuan Anh, Anwitaman Datta
School of Computer Engineering, Nanyang Technological University, Singapore
{ttadinh, anwitaman}@ntu.edu.sg

Abstract

Migration of one's computing infrastructure to the cloud is gathering momentum with the emergence of relatively mature cloud computing technologies. As data and computation are being outsourced, concerns over data security (such as confidentiality, privacy and integrity) remain one of the greatest hurdles to overcome. In the meanwhile, the increasing need for sharing data between or within cloud-based systems (for instance, sharing between enterprise systems or users of a social network application) demands even more care in ensuring data security. In this paper, we investigate the challenges in outsourcing access control of user data to the cloud. We identify what constitute a fine-grained cloud-based access control system and present the design-space along with a discussion on the current state-of-the-art. We then describe a system which extends an Attribute-Based Encryption scheme to achieve more fine-grainedness as compared to existing approaches. Our system not only protects data from both the cloud service provider and unauthorized access from other users, it also moves the heavy computations towards the cloud, taking advantage of the latter's relatively unbounded resources. Additionally, we integrate an XML-based framework (XACML) for flexible, high-level policy management. Finally, we discuss some open problems, solving which would lead to a further robust and flexible cloud-based access control system.

Keywords: *fine-grained access control, Attribute-Based Encryption, XACML, untrusted cloud*

1 Introduction

An enormous amount of data is continuously being generated, with sources ranging from traditional enterprise systems, to social, Web 2.0 applications. It is becoming increasingly common to process data continuously in almost real time as it arrives in streams, in addition to processing (archived) data in batches. Examples of enterprise-scale systems that generate data from their own dedicated sensing infrastructure include stock monitoring,[1] meteorological and environmental monitoring[2] and traffic monitoring [5]. The increased popularity of social and Web 2.0 applications, especially social computing applications like YouTube, Facebook, Twitter, Wikipedia accounts for ever-increasing streams of user-generated content. Finally, ubiquitous

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This work is supported by A*Star grant 102 158 0038.

[1] www.xignite.com

[2] www.noaa.gov

devices such as smart phones equipped with sensing capabilities are driving the growth of other types of social applications such as participatory sensing [15] and personal health monitoring [3].environmental and physiological data.

The variety and abundance of data, coupled with the potential of social interactivity and mash-up services bring sharing of the data to the foreground. For enterprise systems, sharing may translate to new business opportunities, from real-time decision making to smart-city solutions. For social applications, sharing facilitates conformation with the norms or enforcing social bonds. Social computing paradigms such as crowd-sourcing and participatory sensing provide cost-effective alternative services to those using dedicated infrastructure (even when the latter is economically viable).

A critical problem with sharing data is the myriad of security implications. data security has concerned primarily with the question of who gets access to which data. However, the focus has shifted to the question of what aspects of the data to share, and under what context the data can be used. The latter is related to *data privacy*. In this paper, we will be focusing on *fine-grained access control*. The increasing volume of structured data (often arriving in stream) and of the applications (social contexts, near real-time decision support, mash-ups, etc.) demand a scalable and finer access control than the simple all-or-nothing binary sharing. In the following, we identify a core list of access control policies which are found useful in many applications (more complex access scenarios can be derived by combining these *primitives*). For simplicity, let us assume the data consists of multiple attributes, each attribute domain being an ordered space.

- **Filtering policy:** grants access to an attribute if a function is evaluated to `true`. For example, to help detect financial fraud, multiple banks may need to share data of their clients' transactions which contain sensitive information. To achieve this, each bank may only need to share the IDs of *abnormal* transactions whose values exceed a certain threshold. Thus, the function would be $f := \texttt{transaction\_value} \geq \theta$ for a threshold value θ.

- **Granularity policy:** grants access to a *noisy* version of an attribute. For example, in a location sharing application, Alice may not want Bob to know her exact location, but reveal only the area she is in (neighborhood, county, state or country). The policy specifies a granularity level g and a function which transforms the data to different levels of granularity.

- **Similarity policy:** grants access to an user if he can provide inputs that are *similar* to the attribute being shared. For instance, in an online trading system, a selling user submits a price for her item and a set of buyers submitting their estimated price for the item. The seller may want to reveal her price only if it is close to the buyer's estimation. This policy must specify a *closeness* distance within which the attribute will be shared.

- **Summary policy:** grants access to the aggregates (or max, min values) over windows of data. For instance, a mobile health application collects user vital sign (heart rate, blood pressure) every minute. An user wishing to share her data for research purposes may want to reveal only the average, maximum and minimum readings per day (or per sliding window of 24 data items). This policy requires a summary function and specification of a sliding window (starting point, window size and advance step).

Enforcing such access control over a proprietary infrastructure is relatively straightforward. However, instead of building a computing infrastructure from scratch, one can now enjoy the instantly available, elastic and virtually unbounded resources offered by cloud computing vendors at competitive prices [4]. Many

[3] `www.zephyr-technology.com`

[4] `https://developers.google.com/appengine` & `http://aws.amazon.com/ec2`

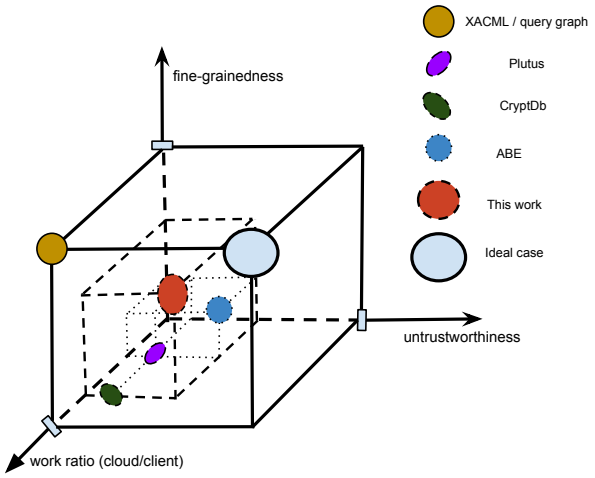

Figure 1: Design space for enforcing access control on the cloud

enterprise systems are thus migrating cloudward, for easy management and maintenance of their infrastructure [14]. At the same time, the easy and instant access to computing resource spawns a plethora small-to-medium size applications to be developed and deployed on the cloud [5]. The result of these trends is that a substantial and increasing amount of data is being hosted and maintained by one (or a small number of) cloud provider. On the one hand, such co-location of data makes it easy to share and perform analytics. On the other hand, the data owner may wish to prevent even its cloud service provider from having access to the data. Therefore, enforcing access control when data is outsourced to the cloud becomes more challenging.

Next we survey the current state-of-the-art in the field of access control on the cloud. We then present a system that achieves the above enumerated fine-grained access control while protecting data from a semi-honest cloud. The system outsources most of the heavy computation to the cloud, and makes use of a standard XML-based framework for high-level access control policy management. Finally, we discuss some open problems that need to be addressed before an *ideal* cloud-based access control system may be realized.

2 State of the Art

A way to characterize any system dealing with access control on a cloud environment would be along the three dimensions depicted in Figure 1. First, the *fine-grainedness* property indicates how many of the fine-grained policies are supported by the access control mechanism. For instance, if the system supports all four policies listed earlier, we can say that it achieves maximum fine-grainedness. On the other hand, if the system only supports an all-or-nothing policy (that means, an user either can access the whole data, or not at all), it is at the minimum with respect to this property. Second, the *untrustworthiness* property determines how untrustworthy the cloud is. If the cloud is completely trusted, this property is at its minimum, whereas if the cloud behaves in completely Byzantine manner, its untrustworthiness is at the maximal level. In between, the cloud may operate at different partially trusted settings. Finally, the *work ratio* property specifies how much work the cloud and the end-user client has to do, in order to carry out the system tasks in relation to each other. The client must be involved to some extent, but if it only has to do minimal, inexpensive work while the cloud carries out all the hard work, this property is at its maximum. In contrast, if the cloud performs very simple tasks such as storage or data distribution, the work ratio is near its minimum. The higher the value of this property is, the more beneficial it is to move to the cloud. In the following, we present the current state of the art, first distinguishing them along the untrustworthiness dimension then discussing how they differ from each other with respect to the other two properties. At this juncture, we

[5] `www.buddypoke.com`

will like to note that the dimensions are not necessarily completely ordered, nor are all relevant points of the dimensions necessarily known. For instance, one may identify other desirable fine-grained access control functionality not enumerated here. Furthermore, some of them may be more important than others. Likewise, different kinds of (mis)behaviors on the cloud service provider's part may have different adversarial effects, that may not all be ordered.

Trusted Cloud. If the cloud is trusted, it is equivalent to when the data owner runs the system on its private infrastructure. This eliminates the need for considering the work ratio property which is maximal because the client can outsource its entire operation to the cloud. The remaining concern is how to maximize the fine-grainedness property.

Traditional relational database systems enforce access control based on view, which essentially pre-computes additional data tables and sets all-or-nothing policies upon them. This approach is static: it requires the data owner to anticipates all possible access scenarios before creating views. For content-based policies such as filtering, where the attribute domain can be very large, this approach is clearly not scalable. Instead, a better practice is to generate views on-the-fly. It is particularly suitable for stream data which can be infinite in size. In particular, Carminati et al. [8] proposes an access control system for the Aurora data model [2] which defines policies as *query graphs* consisting of SQL-like operators that apply to data as it arrives. The authorized user receives the output of the query graph corresponding to the policy he is subject to. Dinh et al. [4] propose a similar approach, but focusing on extending the eXtendible Access Control Markup Language (XACML) framework for easy specification and enforcement of fine-grained policies. It exploits *obligation* elements of XACML, into which any user-defined function can be embedded and executed by the database server before returning data to the requesting user. Wang et al. [19] integrate the two approaches by providing a mechanism to translate the XACML policies into suitable query graphs.

Untrusted Cloud. In most cases, the data owner places great importance in maintaining the confidentiality of the data, either to protect its business asset, the personal or customer data, or to conform with the law. In these cases, it is essential to design safeguards under the assumption that the cloud is *untrusted*: for instance, it could deliberately behave maliciously or it may be infiltrated by a malicious party. A common adversary model for the cloud assumed in existing literature is a *semi-honest* model, i.e. the cloud is not entirely Byzantine.

Plutus [16] and CryptDB [18] are two cloud-based systems dealing with database outsourcing. They assume that the cloud is semi-honest with respect to data storage and retrieval. It means the cloud follows the protocol correctly but it tries to learn the raw data when storing and answering queries. Both systems employ encryption schemes, so that the cloud cannot see the plain-text data. Plutus focuses on revocation and efficiency, to this end it uses a broadcast encryption scheme. CryptDB supports more SQL-like operations such as search, aggregate and join, for which it uses a combination of advanced encryption schemes such as Paillier, order-preserving encryption and proxy re-encryption. These systems, however, support coarse-grained binary access control. In Plutus, the cloud's only job is to store and distribute ciphertexts, hence the work ratio falls heavily on the client side. CryptDb allows the cloud to compute certain operations on ciphertexts while answering user queries, therefore its work ratio is higher.

Attribute-Based Encryption (ABE) schemes [7, 12] allow more fine-grained access control to be enforced on cipher-text. An access tree T is defined over a set of attributes, such that the plaintext can be recovered only if T is evaluated to `true` over some given attributes. Two types of ABE exists: key-policy (KP-ABE) and ciphertext-policy (CP-ABE). They are interchangeable, but the former is more data-centric whereas the latter is user-centric. Essentially, ABE enables one to relax the semi-honest cloud model, i.e. the cloud may try to compromise access control: to grant data access to unauthorized users. ABE ensures that unauthorized access is not feasible nevertheless. ABE can readily be used to enforce filtering policy, therefore it achieves a higher fine-grainedness than CryptDb or Plutus does. Finally, the roles of the cloud in an ABE-based system such as [20] are mainly storage and distribution, therefore the work ratio is similar

to that of [16] ([20] also focuses on access revocation by re-encrypting data).

Summary. Current state-of-the-art systems for cloud-based access control occupy a small zone in the design space, as illustrated in Figure 1, leaving much room for extension. The following section summarizes our proposal [3] that pushes the envelope further, albeit still remaining far from the ideal.

3 Outsourcing Fine-Grained Access Control — The pCloud approach

In the context of the pCloud[6] project, we have proposed an approach [3] that occupies an unique place in the design space (Figure 1). It assumes the same level of cloud trustworthiness as other ABE systems, while achieving higher level of fine-grainedness (as compared to both ABE and CryptDb) and better work ratio (as compared to both ABE and Plutus). The system supports both filtering and summary (sliding window) policies. The former is the result of using KP-ABE, while the latter is achieved by a combination of proxy re-encryption [13] and additive homomorphic encryption. The cloud transforms ABE ciphertexts into more simple Elgamal-like ciphertexts which are cheaper to decrypt. It also computes the sum over ciphertexts without compromising access control, i.e. the users authorized to access the sum only cannot learn the individual data. To achieve access control over sliding windows of size β, we use a set of blind factors $R = \{r_0, .., r_{\beta-1}\}$ when encrypting individual data, then give the authorized user the sum $\sigma = \sum_i r_i$. Using σ, the user can remove the blind factor in the sum of the data, but cannot learn individual r_i necessary for uncovering individual data. We assume that the data domain is small, so that discrete logarithm can be (pre-)computed for all of its values **(as an optimization)**.

KP-ABE: In the KP-ABE scheme, the mesage m is encrypted with a set of attributes A, and the user is given a policy P which is a predicate over a set of attributes $\{a_0, a_1, ..\}$. In addition, the user is given $S = \{s_{a_0}(y), s_{a_1}(y), ..\}$ created by the data owner using a secret sharing scheme such that y can be reconstructed if and only if $P(A) = \texttt{true}$. The scheme guarantees that:

$$\texttt{Dec}(\texttt{Enc}(m, A), P, S) = m \leftrightarrow P(A) = \texttt{true}$$

where $\texttt{Enc}(.)$ and `Dec(.)` are the encryption and decryption function respectively.

Three-phase protocol for sliding window policy: During the ***setup phase***, the data owner creates a secret z, among other things [12]. Suppose the user is given a sliding window policy of size β starting from α. The data owner then computes:

$$\sigma(\alpha, \beta) = \sum_{j=0}^{\beta-1} 2^{\lceil(\alpha+j)/\beta\rceil} R[(\alpha + j)\ mod\ \beta]$$

Notice that for any $k > \alpha$ and $(k - \alpha)\ mod\ \beta = 0$, we have $\sum_{i=k}^{k+\beta-1} 2^{i+k/\beta}.R[(i + k)\ mod\ \beta] = 2^{\frac{k-\alpha}{\beta}}.\sigma(\alpha, \beta)$. Finally, it creates $S = \{s_{a_0}(y/z), s_{a_1}(y/z), ..\}$ for the policy $P := k \geq \alpha$. This way, the valid decryption $\texttt{Dec}(.)$ will return $\phi(z, m)$ for a function ϕ such that m cannot be recovered without knowing z, and $\phi(z_1, m_1).\phi(z_2, m_2) = \phi(z_1.z_2, m_1.m_2)$. The tuple $(z, \sigma(\alpha, \beta), S)$ is sent to the user.

During the ***data outsourcing phase***, the data tuple (k, v_k) where $k = 0, 1, 2..$ is encrypted as:

$$c_k = \texttt{Enc}(g^{v_k + 2^{\lceil k/\beta\rceil}.R[k\ mod\ \beta]}, \mu(k))$$

and sent to the cloud, where g is the generator of a multiplicative group and $\mu(k)$ maps k into a set of attributes.

[6] `http://sands.sce.ntu.edu.sg/pCloud/`

During the ***data streaming phase***, the cloud performs the transformation:

$$t_k = \texttt{Dec}(c_k, P, S) = \phi(z, g^{v+2^{\lceil k/\beta \rceil}.R[k \bmod \beta]})$$

For the i^{th} sliding window, the cloud computes $T_i = \prod_{j=0}^{\beta-1} t_{k+j} = \phi(z^{\beta}, g^{v_k+..+v_{k+\beta-1}+2^i.\sigma(\alpha,\beta)})$ for $k = \alpha + i.\beta$ and sends it to the user. Using z and $\sigma(\alpha, \beta)$, the user can recover $w_i = g^{v_k+..+v_{k+\beta-1}}$ from T_i. Finally, the average for window i^{th} is derived as $avg_i = \frac{\texttt{discreteLog}(w_i)}{\beta}$.

Protocol for filtering policy: The protocols for filtering policies are very similar to that for sliding window. The main differences are that the access policies may involve other conditions besides $\geq$, and that no blind factor is needed since authorized users can access individual data tuples.

Performance. We micro-benchmark our system for both sliding window and filtering policies.[7] The pairing implementation is of type A with 512-bit base field size [1]. The most expensive cryptographic operations are exponentiations and pairings, the latter of which are done by the cloud during transformation. The equality comparison in filtering policies results in 64 pairings, as opposed to the $k \geq \alpha$ conditions in sliding window policy requiring only 1 pairing for large values of k. The transformation takes maximum of $120ms$, while encryption at the data owner takes $179ms$.[8] These latencies are reasonable, because many data streams in practice generate data in intervals of seconds or minutes. Decryption time at the authorized users is around $0.3ms$ - an order of magnitude less than the transformation time at the cloud. This illustrates the benefit of having the cloud performing the heavy computation. The time taken for the setup phase, including pre-computing discrete logarithms for values in $[0, 5000]$ (which is an one-off cost) approximates $0.7s$.

XACML integration. Before starting the transformation operation, the cloud checks if the attributes associated with the ciphertext satisfy the access structure associated with the policy. We integrate XACML framework for systematic managing and matching of policies, so that redundant transformation at the cloud or decryption at the user can be avoided. In our system, the cloud runs an XACML instance, and each data stream is represented as a resource. The data owner defines XACML policies (filtering or summary), for each of which the cloud maintains a list of authorized users. The owner then specifies an XACML request for every ciphertext it sends to the cloud, which is then evaluated against the list of loaded policies. The result is a set of users to whom access to the data should be granted. Finally, the cloud performs transformation on the ciphertexts, or wait until collecting sufficient ciphertexts for the sliding window policies before transforming, and sends the new ciphertexts back to the user. In our micro-benchmark experiments, this additional layer of management incurs small overhead: the time taken for XACML processing with 1500 policies is under $5ms$.

4 Open Problems

Figure 1 hints at the open challenges. When the cloud is not trusted, it will be difficult if not infeasible to achieve the same level of work ratio as systems in the trusted settings can. In order to achieve any level of security with respect to the cloud, the data owner must perform some kind of data encryption. That a stronger level of security requires more expensive encryptions suggests that the more malicious the cloud is, the more work has to be done at the client. This must be taken into consideration when migrating to the cloud, as to balance the saving from outsourcing computation with the overheads of guaranteeing security.

A different level of fine-grainedness. Both filtering and summary policies (as supported in our work) involve simple computations before the data can be returned to the requesting user. These provide a simple data access abstraction, but higher-level abstractions involving more complex computations may be desirable. One example is a policy that grants access only to results of certain data mining algorithms or statistical

[7] The source code is available at `code.google.com/p/streamcloud`

[8] Experiments are run o desktop machines having 2 2.66Ghz DuoCore with 4GB of RAM

functions. When the cloud is trusted, this can be accomplished by extending XACML-based systems. In untrusted clouds, this must be accomplished using homomorphic encryptions. Although fully homomorphic encryption schemes are feasible in theory, making them practical for complex functions remain a challenge.

Decision support for determining sharing policies. So far, we have been assuming that the data owner knows what data is sensitive and what policies are to be chosen. In reality, these decisions are not obvious to make. For instance, many systems that collect user data and publish the anonymized versions have run into public relation disaster (Netflix and AOL, for example) because data can be linked to reveal sensitive information. Furthermore, the complexity and dynamics of social networks make it difficult for users of such systems to determine which policies to set for which friends [9]. *Differential privacy* can help reason about the former problem, as it provides a bound on how much privacy leakage would incur if the user decided to share something. A recent work [10] shows that the cloud can be delegated to ensure differential privacy on ciphertext, but it uses a encryption scheme which does not provide fine-grained access control. Likewise, decision support mechanisms to determine what to share with whom based on social and trust relations, as well as approaches to detect and prevent leakage of information are essential missing pieces.

Mitigating proactively malicious cloud. All systems covered in this article stop short of considering a fully malicious cloud. They assume the cloud adversary model to be semi-honest, i.e. it will execute the protocols truthfully while trying to compromise some security properties. Such an assumption does not suffice once the cloud has incentives to actively skip or subvert the delegated computation. Skipping computation may be driven by economic interest (the cloud doing less work while still charging the users), while competition may be a reason to distort computations. Both data owners and end users must be able to reliably detect such behaviors. This is a special case of verifiable computation, in which the client is able to verify the output of a function it outsourced to a third party. It has been shown that any computation can be outsourced with guaranteed input and output privacy [11], but the existing protocols are inefficient. A more practical approach may be probabilistic in nature. Other alternatives may include dividing the data or computation task over multiple cloud service providers, if the independence of these providers (alternatively, prevention of their collusion) can be guaranteed.

Other open problems. Even for the approach presented in Section 3, there are several unaddressed issues. We have not considered access revocation, which in our case requires the data owner to change the attribute set during encryption. We plan to investigate if existing revocable KP-ABE schemes [6] can be integrated, especially if they allow revocation to be outsourced to an untrusted cloud. Other interesting extensions are to add support for policies with negative attributes [17], and for encryption with hidden attributes. The former allows for a wider range of access policies, whereas the latter provides attribute privacy which is necessary when encryption attributes are the actual data.

5 Conclusions

In this article, we have discussed important challenges in designing a cloud-based fine-grained access control system. We have identified a core set of fine-grained access policies that are desirable in many real-life applications. Existing systems supporting this full set of policies assume a trusted cloud. For untrusted cloud, current state-of-the-art systems share similar semi-honest adversary models, while they differ in the levels of fine-grainedness in access control and the work ratio between the cloud and the client. We summarize our recent work that pushes the envelope [3] further. Finally, we outline a number of open problems that need to be overcome so that we can achieve, or at least get closer to the ideal cloud-based access control system.

References

[1] Key-policy attribute-based encryption scheme implementation. `http://www.cnsr.ictas.vt.edu/resources.html`.

[2] Daniel J. Abadi, Don Carney, Ugur Cetintemal, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael stone-

braker, Nesime Tatbul, and Stand Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–39, 2003.

[3] Dinh Tien Tuan Anh and Anwitaman Datta. Stream on the sky: Outsourcing access control enforcement for stream data to the cloud. *arXiv:1210.0660*, 2012.

[4] Dinh Tien Tuan Anh, Wang Wenqiang, and Anwitaman Datta. City on the sky: extending xacml for flexible, secure data sharing on the cloud. *Journal of Grid Computing*, pages 151–172, 2012.

[5] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: a stream data management benchmark. In *VLDB*, pages 480–91, 2004.

[6] Nuttapong Attrapadung. Revocation scheme for attribute-based encryption. RCIS Workshop, `http://www.rcis.aist.go.jp/files/events/2008/RCIS2008/RCIS2008_3-5_Nuts.pdf`, 2008.

[7] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–34, 2007.

[8] Barbara Carminati, Elena Ferrari, and Kian Lee Tan. Enforcing access control over data streams. In *SACMAT*, pages 21–30, 2007.

[9] Gorrell P. Cheek and Mohamed Shehab. Policy-by-example for online social networks. In *SACMAT*, pages 23–32, 2012.

[10] Ruichuan Chen, Alexey Reznichenko, and Paul Francis. Towards statistical queries over distributed private user data. In *NSDI*, 2012.

[11] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO'10*, August 2010.

[12] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS'06*, pages 89–98, 2006.

[13] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *20th Usenix conference on Security*, 2011.

[14] Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In *SIGCOMM*, 2010.

[15] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Gorackzko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: a distributed mobile sensor computing system. In *4th international conference on embedded networked sensor systems*, 2006.

[16] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: scalable secure file sharing on untrusted storage. In *FAST 2003*, 2003.

[17] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *CCS'07*, pages 195–203, 2007.

[18] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: a practical encrypted relational dbms. Technical Report MIT-CSAIL-TR-2011-005, CSAIL, MIT, 2011.

[19] Wen Qiang Wang, Dinh Tien Tuan Anh, Hock Beng Lim, and Anwitaman Datta. Cloud and the city: Facilitating flexible access control over data-streams. In *SDM*, 2012.

[20] Schucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable and fine-grained data access control in cloud computing. In *INFOCOM*, pages 534–542, 2010.