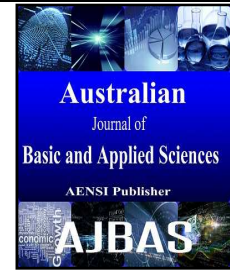




AUSTRALIAN JOURNAL OF BASIC AND APPLIED SCIENCES

ISSN:1991-8178 EISSN: 2309-8414
Journal home page: www.ajbasweb.com



Cost Effective Management of Intermediate Datasets in Cloud Environment

Sarah Prithvika P.C., Kalai Arasi S., Angel Princes A. and Ramani S.

Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India

Address For Correspondence:

Sarah Prithvika P.C., Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, Tamil Nadu, India.
E-mail: sarah.chandran@gmail.com

ARTICLE INFO

Article history:

Received 10 December 2015

Accepted 28 January 2016

Available online 10 February 2016

Keywords:

Cloud computing; security; deduplication; resource sharing.

ABSTRACT

Cloud computing technology helps users to share resources. Cloud computing provides good computing power, reasonable cost of services, scalability, high performance, accessibility and availability. The cost of service is based on the pay as per usage scheme. Companies can scale up or scale down based on their business requirements. In some cases it is cost-effective to save the intermediate datasets that are produced when data is processed. Communities interested in a particular resource, may share the stored information. When resources are shared, security concerns arise. There is also a need to store the data efficiently as storage cost depends on the amount of data stored. This paper deals with storing the intermediate datasets securely and cost-effectively.

INTRODUCTION

When data is processed, intermediate datasets are produced; these data sets are used by some communities for research. So, the intermediate data sets can be stored in the cloud environment and can be shared by authorized people. This is especially cost-effective in cases where generating the intermediate datasets repeatedly is costly. The intermediate dataset is shared only with authorized persons, but when this data is available in the cloud environment, sensitive information may be leaked to adversaries. In order to overcome this problem, encryption is done. Encrypting all the datasets may not be a good idea, as each time the data has to be decrypted. Instead, as mentioned in (Zhang, X., 2013), some of the intermediate datasets can be encrypted and the others can be unencrypted while still maintaining reasonable security. Duplicate data stored in cloud must be eliminated to save storage cost. Data deduplication is the process of ensuring that duplicate copies of the data don't exist in the cloud. But, performing data deduplication of encrypted data on the cloud is challenging. Traditionally, convergent encryption is used, but there are some drawbacks in this type of encryption.

Related Work:

There has been a lot of research in the areas of intermediate dataset storage and deduplication. Roy *et al.* (2010) analyzed the data privacy issues caused by MapReduce and designed a system named Airavat which uses mandatory access control. Policies are controlled by data providers, based on which data can be accessed. Puttaswamy *et al.* (2011) explain about Silverline, which is a set of tools that identifies and encrypts the functionally encryptable data without affecting the application on the cloud. Zhang *et al.* (2011) discuss about a system named Sedic which automatically partitions computing jobs based on the security labels assigned to the data and puts the part without sensitive information to a public cloud. Valentina Ciriani *et al.* (2010), propose a technique in which encryption and fragmentation of data are done to ensure privacy of sensitive data. The sensitive associations among attributes are removed. Davidson *et al.* (2011) propose techniques to achieve module level privacy preserving based on workflow provenance. The minimum data that must be hidden in

Open Access Journal

Published BY AENSI Publication

© 2016 AENSI Publisher All rights reserved

This work is licensed under the Creative Commons Attribution International License (CC BY).

<http://creativecommons.org/licenses/by/4.0/>



Open Access

To Cite This Article: Sarah Prithvika P.C., Kalai Arasi S., Angel Princes A. and Ramani S., Cost Effective Management of Intermediate Datasets in Cloud Environment. *Aust. J. Basic & Appl. Sci.*, 10(1): 232-237, 2016

order to maintain security is identified. X. Zhang *et al.* (2013) propose a privacy leakage constraint-based method to identify which intermediate data sets must be encrypted, so that the cost of preserving privacy can be saved while still meeting the privacy requirements of data holders. In the proposed scheme we combine dataset encryption and deduplication to cost-effectively manage data.

- **Background:**

- A. **Convergent Encryption:**

Using convergent encryption two matching plaintext files produce the same ciphertext. It is also known as content hash keying. This is used in cloud computing to remove duplicate files from cloud storage without the provider possessing the encryption keys. Convergent encryption is susceptible to an attack that confirms if a file exists. An attacker can find out if a target has a certain file by encrypting a plain-text file and then he can compare the output with files in possession by the target. This attack poses a problem for a user saving information that is not distinct, i.e. also either available in public or already possessed by the adversary. For eg., books that have been banned or files that cause a copyright infringement. It can be said that a confirmation of a file attack can be averted by adding a unique piece of data, like a few characters to the plain text before encryption; this will make the uploaded file to be unique and therefore it produces a unique encrypted file.

An attacker can learn the missing information from a file. This attack is described by Drew Perttula in 2008. This type of attack applies to the encrypted files that are only a slight modification of a public document. For example, if a person encrypts a bank form including the bank account number, then an attacker who knows the generic bank form format may find the person's bank account number by generating bank forms for all bank account numbers, encrypting them and then doing a comparison of those encryptions with the person's encrypted file. Note that this attack can be used to attack a large number of targets at once, and the presence of this problem can be used in any type of form document like income tax returns, financial details documents, healthcare management forms, employment forms, etc. There is no known method for decreasing the intensity of this attack.

- B. **Data Duplication:**

Data Deduplication is used to improve the utilization of storage space and can be used to transfer data across the network to reduce the number of bytes that must be sent across the network. In the deduplication process, distinct pieces of data or patterns of bytes are identified and stored during the analysis process. A comparison is made between a chunk of data and the stored copy. If a match is found, then the duplicate chunk is replaced with a reference that refers to the stored chunk. Suppose the same set of bytes occurs several times (the match frequency depends on the chunk size), the data that must be stored on cloud or transferred can be greatly reduced. In client side deduplication, the data is deduplicated before it is transferred to the storage area. The client processes the data before sending it across the network. Server side deduplication is where the clients pass all of their data over to the network and once all data is transferred, the data is processed and duplicate data is identified and deleted. Even though more data is sent across the network in this method, the workload of the client is reduced. For client side deduplication, we need the right amount of disk, memory and CPU power. The clients take more workload, but it reduces the load of the network. However, the workload it places on the client could affect the applications that are running on the client. When network resources are limited and when the client has the capacity in terms of memory and processor power to devote to this extra process, then client side deduplication can be used. Data deduplication is one of the most widely used technologies in storage right now as it helps companies to save money. If deduplication of data is done, the existing storage space can be utilized in a better fashion, which can save money by using the resources at hand more efficiently. If less data is stored then the amount of data in backup is also less, which again means less hardware and backup media. If less data is stored, then less data has to be sent over the network in case of an emergency, which means money in hardware and network costs over time are saved. The advantages of data deduplication are reduced hardware and backup costs, increased storage and network efficiency and reduced costs for business continuity / disaster recovery. Deduplication is different from compression which is another technique used for decreasing storage requirements. Deduplication eliminates redundant data whereas compression makes use of algorithms to save data more concisely. Compressions may be lossy or lossless. Compression is lossless when no data is lost in the process. Compression is lossy when some data is lost in the process. It is frequently used with audio and video files. It actually deletes some of the data included in a file to save space. Deduplication removes extra copies of data; the data is not lost. Compression does not delete duplicated data; the storage system may continue to have many copies of compressed files. In simplified terms, data deduplication compares data and removes copies that are already present in the data set. In deduplication, removal of blocks that are not distinct is done. The process consists of four steps:

1. The input data is broken down into blocks or "chunks."
2. A hash value is calculated for each block of data.
3. A check is done to locate if another block of the same data has already been stored.

4. The duplicate data is replaced with a reference to the object already present in the database.

Once the data is broken into chunks, an index can be generated, and the duplicates can be found and removed. Only one copy of every chunk is stored.

Proposed Scheme:

Privacy issues caused by saving intermediate datasets in cloud are important. Storage of intermediate data must be done economically. So we propose a solution for an organization to save intermediate data economically and securely in the cloud. Encrypting all the datasets is expensive, so based on, only some of the intermediate datasets are encrypted while still maintaining privacy. If an adversary can find the sensitive information based on combined information from datasets d_1 and d_2 as shown in Table. 1 and Table. 2, then data leakage can be prevented by encrypting either d_1 or d_2 . Consider the following datasets:

Table I: Data set d_1 with the patient's contact information.

Patient Id	Patient Name	Phone No.	Address
34999	Esther, Susan	839292929	No.8, Trunk Road, Seattle, USA.
35000	Thomas, Steve	847473636	No. 9, Hill Road, Washington, DC

Table II: Data set d_2 listing patient's illness.

Patient Id	Illness
34999	HIV
35000	Tuberculosis

Using both data sets it is easy to infer the identity and illness of the patient. So we have to encrypt one of the data sets to ensure privacy. Now, another user in the organization may upload the same intermediate data set. So deduplication of intermediate data set is done to avoid space wastage before upload. Let d_0 be a privacy-sensitive actual data set. Let $D = \{d_1, d_2, \dots, d_n\}$ denote a group of intermediate data sets produced from d_0 , here n denotes the number of intermediate data sets. Intermediate data set consists of both intermediate and resultant data sets. Intermediate data is managed using data provenance like in (Yuan, D., 2011). Provenance is the origin or source of derivation of data. Thus it becomes possible to produce a data set from its nearest available predecessor rather than create them all over again (Yuan, D., 2011; Davidson, S.B., 2011). Directed Acyclic graphs (DAG) are used to denote the relationship between the data sets.

It also indicates how a data set was generated. A DAG describing the generation relationships of intermediate data sets D from d_0 is called Sensitive intermediate data set graph (SIG). SIG is transformed into a Sensitive Information Tree (SIT) if each of the data sets is produced from only one parent data set. An attribute vector is used to set important properties of the data set d_i . The vector is represented as $\langle S_i, \text{Flag}_i, f_i, \text{PL}_i \rangle$. S_i represents the size of data set d_i . Flag_i indicates if the data set d_i is hidden. If a dataset is hidden, then it has to be encrypted/decrypted each time it is processed. f_i represents the frequency of accessing or processing d_i . We have to choose the dataset with less cost. If f_i is large and if it is hidden, then more cost is incurred as it would involve frequent encryption and decryption. PL_i is the privacy leakage through d_i and is computed by $\text{PL}_s(d_i)$. The prices of various services required by en/decryption is combined into one, in order to focus on the main idea rather than go into the pricing details. This combined price is indicated by PR . Data sets in D can be split into two. D^{enc} denotes encrypted data sets. D^{unc} denotes unencrypted data sets. The equations $D^{\text{enc}} \cup D^{\text{unc}} = D$ and $D^{\text{enc}} \cap D^{\text{unc}} = \emptyset$ hold. We can describe this problem as an optimization problem. The goal must be to minimize the following:

$$\text{CR}_{\text{pp}} = \sum_{d_i \in D^{\text{enc}}} S_i \cdot \text{PR} \cdot f_i, \quad D^{\text{enc}} \subseteq D. \quad (1)$$

The privacy leakage caused by the data that is not encrypted data should be within a given threshold. Let the privacy leakage threshold allowed by data holder be denoted as ϵ . The privacy leakage constraint can be defined as $\text{PL}_m(D^{\text{unc}}) \leq \epsilon$, $D^{\text{unc}} \subseteq D$. It is difficult to find the exact value of $\text{PL}_m(D^{\text{unc}})$, so PLC will be replaced with one of its sufficient conditions. Let d_u and d_v be two data sets and let the respective privacy leakages be $\text{PL}_s(d_u)$ and $\text{PL}_s(d_v)$. The joint privacy leakage caused by both is $\text{PL}_m(\{d_u, d_v\})$. Since information gain will never be negative, $\text{PL}_m(\{d_u, d_v\})$ will not become more than the sum of $\text{PL}_s(d_u)$ and $\text{PL}_s(d_v)$, i.e. we can say that $\text{PL}_m(\{d_u, d_v\}) \leq \text{PL}_s(d_u) + \text{PL}_s(d_v)$, the equality holds if and only if there is no overlap in the information provided by d_u and d_v . Hence, the sum of privacy leakage of unencrypted data sets can be taken as upper bound of $\text{PL}_m(D^{\text{unc}})$.

C. Privacy-Preserving Cost Reducing Heuristic Algorithm:

Using this algorithm, a solution is found from a limited search space. Heuristic functions are used to get heuristic values. A heuristic function $f(\text{SN}_i)$, is used to find the heuristic value of SN_i . $f(\text{SN}_i)$ is made up of two components of heuristic information, i.e., $f(\text{SN}_i) = g(\text{SN}_i) + h(\text{SN}_i)$, where $g(\text{SN}_i)$ is the information gained between the start state and the current state node SN_i , and the information $h(\text{SN}_i)$ is computed from the current state node to the goal state, respectively. The algorithm aims to fetch the data sets with small cost but high privacy leakage to encrypt. $g(\text{SN}_i)$ is defined as follows:

$$g(SN_i) \triangleq C_{cur} / (\varepsilon - \varepsilon_{i+1}). \quad (2)$$

The notation C_{cur} is the privacy-preserving cost that has culminated thus far, ε is the initial privacy leakage threshold, and ε_{i+1} is the privacy leakage threshold for all the layers after L_i . C_{cur} is computed by the following formula:

$$C_{cur} = \sum_{d_j \in \cup_{k=1}^i ED_k} (S_j \cdot PR \cdot f_j). \quad (3)$$

If C_{cur} is small, it means that the total privacy-preserving cost also will be small. If $(\varepsilon - \varepsilon_{i+1})$ is large, it means more data sets before L_{i+1} are unencrypted based on the RPC property, i.e., the amount of privacy-preserving expense saved is more. The value of $h(SN_i)$ is computed using $h(SN_i) = (\varepsilon_{i+1} \cdot C_{des} \cdot BF_{AVG}) / PL_{AVG}$. Similar to $(\varepsilon - \varepsilon_{i+1})$ in the definition of $g(SN_i)$, smaller ε_{i+1} in $h(SN_i)$ means more data sets before L_{i+1} remain unencrypted. If in an SIT, a data set with less depth is encrypted, then it means that a lot of data sets are possibly unencrypted than that with more depth, because the former has the possibility of possessing more descendant data sets. For a state node SN_i , its corresponding ED_k has data sets that are the roots of a variety of subtrees of the SIT. These trees make a forest, denoted as F_{π_i} . In $h(SN_i)$, the total cost of the data sets in F_{π_i} is represented as C_{des} and is computed using the formula:

$$C_{des} = \sum_{d_1 \in ED_k} \sum_{d_j \in PD_{d_1}} (S_j \cdot CR \cdot f_j). \quad (4)$$

The lesser the value of C_{des} , it means fewer data sets in the following layers will be encrypted. The notation BF_{AVG} denotes the average branch factor of the forest F_{π_i} , and can be calculated by $BF_{AVG} = \frac{N_E}{N_I}$, where N_E denotes the number of edges and N_I denotes the number of internal data sets in F_{π_i} . The goal state is used to derive the privacy-preserving solution and corresponding cost. A small BF_{AVG} means that the search space for sequent layers will also be small. So this helps in finding a near-optimal solution faster. The value of PL_{AVG} denotes the average privacy leakage of data sets in F_{π_i} , calculated by the following formula:

$$PL_{AVG} = \sum_{d_1 \in ED_k} \sum_{d_j \in PD_{d_1}} PL_S(d_j) / N_I \quad (5)$$

The objective is to encrypt data sets that disclose more sensitive information while at the same time have less cost. So, a higher value for PL_{AVG} means that more data sets in F_{π_i} should be encrypted to preserve privacy. The heuristic value of the search node SN_i can be found using the following formula:

$$f(SN_i) = C_{cur} / (\varepsilon - \varepsilon_{i+1}) + (\varepsilon_{i+1} \cdot C_{des} \cdot BF_{AVG}) / PL_{AVG} \quad (6)$$

Based on this, a privacy-preserving cost reduction algorithm is developed as shown in Algorithm 1. The concept is that the algorithm iteratively selects a state node with the highest heuristic value. After finding such a state node, it extends its child state nodes until it reaches a goal state node. Algorithm 1 gives the details of the heuristic algorithm. A priority queue maintains the state nodes. A state node is added to the priority queue only if it qualifies. To ensure that the size of the priority queue does not increase dramatically, the algorithm keeps only the state nodes with top K highest heuristic values. A local encryption solution is first derived from CDE_i before determining to add child search nodes in layer L_{i+1} into the priority queue. The algorithm has the disadvantage because it has to check all combinations of data sets in CDE_i , so it becomes less efficient. To circumvent this problem, the algorithm ascendingly sorts the data sets in CDE_i based on the value of $\frac{C_k}{PL_S(d_k)}$, where $d_k \in CDE_i$ and $C_k = S_k \cdot PR \cdot f_k$. In case $|CDE_i|$ exceeds a threshold M, then only the first M data sets in the sorted CDE_i will be analyzed while the remaining are set to be encrypted. The data sets that will remain unencrypted will be those with less privacy leakage and high cost for preserving privacy. The value of $\frac{C_k}{PL_S(d_k)}$ can be helpful in finding the needed data sets with higher possibility. So, the algorithm is used to reach the goal state in the state space as close as possible. SORT and SELECT are two external functions.

SITs cannot be suitable for all situations, so in some cases this approach has to be extended to the SIG, as sometimes an intermediate data set can be derived from more than one parent data set. Let d_m indicate a data set that gets data from more than one predecessor. Since only one dataset is assumed to exist in an SIG, all the data paths from the root data set d_0 to d_m must meet at one point other than d_m itself. Let d_s denote this source data set, e.g., $d_s = d_0$. The inequality $PL_S(d_m) \leq PL_S(d_s)$ holds because all the sensitive information of d_m comes from d_s . The Root Privacy Coverage (RPC) property says that if an intermediate node in an SIT is unencrypted then it is unnecessary to check the posterity data sets of that intermediate node and such a subtree is called unencrypted subtree, indicated by UST. The Encrypted Data Set Tree (EDT) property says that all encrypted data sets in an SIT form a tree structure if the privacy-preserving cost is as less as possible. If all data sets in $PD_i(d_s)$ are unencrypted, then all the predecessors of d_m after the layer L_i will not be encrypted according to RPC property. If all data sets in $PD_i(d_s)$ are encrypted, then if d_m is a child of data set in $PD_i(d_s)$, d_m is added to CDE_{i+1} for the subsequent round. Let d_p be the parent dataset. Then, the edges pointing to d_m from parents except d_p are deleted. Logically we can say that d_m is compressed candidate data set of several imaginary data sets in CDE_{i+1} . If part of data sets in $PD_i(d_s)$ are encrypted while other data sets are

unencrypted, then as per the RPC property, it is considered safe to expose part of sensitive information from the unencrypted data set.

Algorithm1. Cost effective and privacy-preserving heuristic:

Description	Identifies the intermediate datasets that need to be encrypted
Input	A SIT with root d_0 . All attributes values of every intermediate dataset are given.
Output	A vector of solutions $\langle \pi_1 \dots \dots \pi_H \rangle$ that comprise a privacy-preserving solution; and the privacy-preserving cost.
Step 1	Initialize the following variables 1.1 Define a priority queue: PQueue. 1.2 Construct the initial search node: $SN_0 = \langle \langle \pi_0 \rangle \langle \{d_0\}, \emptyset \rangle, f(SN_0) \leftarrow 0, ED_0 \leftarrow \{d_0\}, C_{cur} \leftarrow 0, \epsilon_1 \leftarrow \epsilon \rangle$, i.e. the parameters are current solution, the current heuristic value, the current ED, the current cost and the privacy leakage requirement for the subsequent layers. 1.3 The nodes are added into PQueue: $PQueue \leftarrow SN_0$
Step 2	Retrieve the search nodes and add their child search nodes to PQueue. 2.1 Fetch the search node with the highest heuristics from PQueue: $SN_i \leftarrow PQueue$. 2.2 If $ED_i = \emptyset$, then a solution is found and the algorithm will go to Step 3. 2.3 The datasets in CDE_i are labeled as encrypted if their privacy leakage is larger than ϵ_i . The unlabeled datasets in CDE_i are sorted ascendingly based on $C_k / PL_s(d_k)$, $d_k \in CDE_i$: $SORT(CDE_i)$. Only the first M datasets are will be considered to generate the candidate nodes. 2.4 Generate all the possible local solutions in Λ_i . 2.5 Select a solution from Λ_i : $\pi \rightarrow SELECT(\Lambda_i)$: a) Find out the privacy leakage upper bound of this solution along with the encryption $cost: PL_{local} \leftarrow \sum_{d \in UD_\pi} PL_s(d), C_{local} \leftarrow \sum_{d_k \in ED_\pi} (S_k \cdot CR \cdot f_k)$, where $\pi = \langle ED_\pi, UD_\pi \rangle$ b) Calculate the remaining privacy leakage $\epsilon_{i+1} \leftarrow \epsilon_i - PL_{local}$. 2.6 Calculate the heuristic value 2.7 Construct new search node from the obtained values and add it to PQueue. Then goto Step 2.1.
Step 3	Obtain the global encryption cost C_{global} : $C_{global} \leftarrow C_{cur}$, and the solution $\langle \pi_1, \dots, \pi_H \rangle$.

B. Encryption:

Using convergent encryption two matching plaintext files produce the same ciphertext. So, it is susceptible to some attacks that can be used to find if a file exists or it can be used to find the remaining information. We assume a limited set of users like members of a particular scientific community in our system. The role of the user is limited to splitting files into blocks, encrypting them with the convergent encryption technique and secret key. We can create a secret value based on and mix that value into the encryption key (so instead of $symmetric_key = H(plaintext)$, we have $symmetric_key = H(added_secret_key, plaintext)$). By adding a secret to the symmetric encryption key, the scope of users is limited. Attackers who are not in this group of users will not be able to use the learn-the-remaining-information attack, nor can they use the confirmation-of-a-file attack. Thus identical plain text will produce identical cipher text and thereby enable block level deduplication at cloud storage while still avoiding learn-the-remaining-information attack and the confirmation-of-a-file attack.

Experimental Analysis:

Encrypting all the data sets for privacy preserving is used in many places; let it be denoted as ALL_ENC. The privacy preserving cost of ALL_ENC is denoted as C_{ALL} . C_{ALL} can be calculated using the formula:

$$C_{ALL} = \sum_{d_k \in D} S_k \cdot PR \cdot f_k \tag{7}$$

We can say that the cost of encrypting few datasets is lesser than the cost of encrypting all datasets. Experiments have also been conducted to indicate the same. Let privacy preserving cost be denoted as C_{HEU} , it can be calculated using the following formula,

$$C_{HEU} = \sum_{d_k \in ED} S_k \cdot PR \cdot f_k \tag{8}$$

Let C_{SAV} denote the difference between C_{ALL} and C_{HEU} . The difference C_{SAV} becomes larger when the number of intermediate data sets increases. The expense can be reduced when the number of data sets increases.

Conclusion And Future Work:

This paper discusses about combining convergent encryption and data deduplication for cost-effective intermediate data management in cloud environment. In this scheme, deduplication will be applied only to the files of those users that share the secret encryption key thus dramatically limiting deduplication effectiveness.

Future work can be done to improvise on this issue. Column-wise encryption can be explored to see if it offers less overhead than encrypting the entire dataset.

REFERENCES

- Zhang, X., C. Liu, S. Nepal, S. Pandey, J. Chen, 2013. "A Privacy Leakage Upper Bound Constraint-Based Approach for Cost-Effective Privacy Preserving of Intermediate Data Sets in Cloud," IEEE Transactions On Parallel And Distributed Systems, 24-6.
- Roy, I., S.T.V. Setty, A. Kilzer, V. Shmatikov, E. Witchel, 2010. "Airavat: Security and Privacy for Mapreduce," Proc. Seventh USENIX Conf. Networked Systems Design and Implementation (NSDI'10), 20.
- Puttaswamy, K.P.N., C. Kruegel, B.Y. Zhao, 2011. "Silverline: Toward Data Confidentiality in Storage-Intensive Cloud Applications," Proc. Second ACM Symp. Cloud Computing (SoCC '11).
- Zhang, K., X. Zhou, Y. Chen, X. Wang, Y. Ruan, 2011. "Sedic: Privacy-Aware Data Intensive Computing on Hybrid Clouds", Proc. 18th ACM Conf. Computer and Comm. Security (CCS '11), 515-526.
- Ciriani, V., S.D.C.D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, P. Samarati, 2010. "Combining Fragmentation and Encryption to Protect Privacy in Data Storage," ACM Trans. Information and System Security, 13(3): 1-33.
- Davidson, S.B., S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen, 2011. "On Provenance and Privacy," Proc. 14th Int'l Conf. Database Theory, 3-10.
- Davidson, S.B., S. Khanna, T. Milo, D. Panigrahi, S. Roy, 2011. "Provenance Views for Module Privacy," Proc. 30th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS '11), 175-186.
- Davidson, S.B., S. Khanna, V. Tannen, S. Roy, Y. Chen, T. Milo, J. Stoyanovich, 2011. "Enabling Privacy in Provenance-Aware Work-flow Systems," Proc. Fifth Biennial Conf. Innovative Data Systems Research (CIDR '11), 215-218.
- Drew Perttula and Attacks on Convergent Encryption https://tahoe-lafs.org/hacktahoelafs/drew_perttula.html
- Yuan, D., Y. Yang, X. Liu, J. Chen, 2011. "On-Demand Minimum Cost Benchmarking for Intermediate Data Set Storage in Scientific Cloud Workflow Systems," J. Parallel Distributed Computing, 71(2): 316-332.
- Muniswamy-Reddy, K.K., P. Macko, M. Seltzer, 2010. "Provenance for the Cloud," Proc. Eighth USENIX Conf. File and Storage Technologies (FAST '10), 197-210.
- Perttula, Attacks on convergent encryption. <http://bit.ly/yQxyvl>.