Model of an Encrypted Cloud Relational Database Supporting Complex Predicates in WHERE Clause

Sidorov, Vasily; Ng, Wee Keong

2014

Sidorov, V., & Ng, W. K. (2014). Model of an Encrypted Cloud Relational Database Supporting Complex Predicates in WHERE Clause. 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), 667-672.

https://hdl.handle.net/10356/83218

https://doi.org/10.1109/CLOUD.2014.94

Model of an Encrypted Relational Cloud Database Supporting Complex Predicates in WHERE Clause

Vasily Sidorov
School of Computer Engineering
Nanyang Technological University
Singapore
Email: vasily001@e.ntu.edu.sg

Wee Keong Ng
School of Computer Engineering
Nanyang Technological University
Singapore
Email: wkn@pmail.ntu.edu.sg

***Abstract*—Even though the concept of a Database-as-a-Service (DaaS) is becoming more popular and offers significant expenditure cuts, enterprises are still reluctant to migrate their data storing and processing to the cloud. One of the reasons to that is a lack of solid security guarantees. Encrypted database is one of the major approaches to address the security of cloud data processing. However, in order to provide processing capabilities over encrypted data, multiple techniques need to be combined and adjusted to work together. This paper introduces a modular and extensible framework model of an encrypted database, which makes it possible to execute a wide range of queries, including those with complex arithmetic expressions, retaining data privacy even with an adversary gaining full access to the database server. Proposed model could be used as a basis for encrypted database systems with various functional requirements.**

***Keywords*-cloud database security; querying encrypted data; complex query predicates;**

I. Introduction and Related Work

Cloud data storage and processing in general, and DaaS in particular are becoming more and more available and financially attractive to the enterprises. It gives them an opportunity to drastically reduce costs of storing and processing data through all the benefits that a cloud computing paradigm introduces.

However, data security risks leave many of them reluctant to migrating to the cloud. Risks of data theft are increasing, and in addition to the third-party adversaries the cloud platform provider itself could be considered an ill-wisher. As long as the cloud platform has full access to the data it stores, it should be considered at least as a passive threat, i.e., an eavesdropper.

In order to secure the privacy of the data an encryption could used but that immediately makes data processing in the cloud impossible without additional efforts. Processing of the encrypted data in the cloud raises many research issues, e.g., how to perform OLAP/OLTP operations over the encrypted data, how to search through the encrypted data, what should be done in case en-/decryption keys are compromised, how to deal with multi-user scenarios, etc.

Until there is a practically usable fully homomorphic encryption, one of the most challenging problems is performing arbitrary or at least a wide set of operations over the remote encrypted data. In 2009 Craig Gentry had published his thesis that described a fully homomorphic encryption scheme [1]. However, it appeared to be too resource consuming to be used in practice. Later there was an attempt to make it more practical and some results were achieved, but still it was distant from being ready for practical use [2]. A simplified version of Gentry's scheme, so-called "a somewhat-homomorphic encryption" was used in the context of an encrypted database, but since it lacks fully homomorphic properties it is unable to perform arbitrary operations [3]. As of now there are no known practically usable fully homomorphic encryption schemes, which makes the research in this area demanded.

One of the most fundamental works on querying an encrypted storage of structured data was done by H. Hacıgümüş *et al.* in [4]. This work was among the first to fully cover an implementation of an encrypted relational database. Paper introduces a very important to the future research idea of storing additional data along with the encrypted data in order to facilitate the search. Paper suggests to encrypt the data at a tuple level and makes it possible to use a predefined set of attributes in queries, including an ability to use *all* attributes.

The ideas proposed by H. Hacıgümüş *et al.* experienced an evolutionary development in later works by Z. Yang *et al.* [8], who suggested a specific way to store redundant data for searching purposes along with the ciphertexts; and B. Hore *et al.* [9], where they extended the model of H. Hacıgümüş *et al.* and added range queries to it. Our previous works [10], [11] were in some measure extending these ideas and created a basis for our research in the direction of generalization of these models, which is reflected in this paper.

Another important work on encrypted databases is the CryptDB, which is being developed in the MIT Computer Science and Artificial Intelligence Laboratory by R. Popa *et al.* [5]–[7]. CryptDB introduces an approach

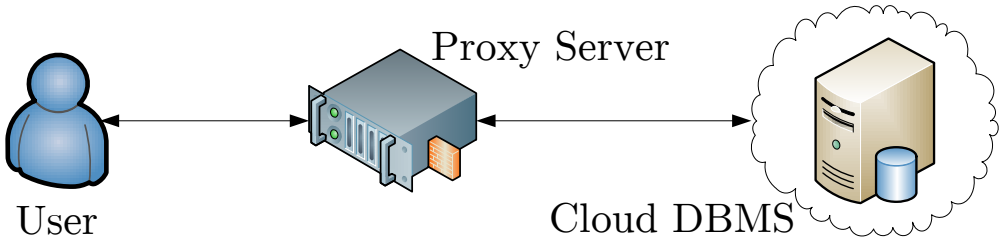

Figure 1: Model architecture

called "*onion* encryption" — a multi-layered encryption scheme with the server "peeling" outer layers when needed. While CryptDB is one of the most developed encrypted DBMSs, its security is based on discussable assumptions (e.g., at some points of the query execution an untrusted server can receive the secret en-/decryption keys), and its functionality lacks some of the useful features (e.g., there is no support for queries containing multiplication, relational joins are rather limited, and some other limitations). One of the purposes of this work is to address these issues. We aim to provide a wider functionality and introduce a more severe threat model, which, in our opinion, better reflects the real conditions of using a cloud platform provided by a third party. As an example, the model described in this work makes it possible to run a query like `SELECT * FROM t1, t2 WHERE t1.a = t2.b AND t1.a + t2.c * t1.d > t2.e` against arbitrary tables `t1` and `t2`, which the CryptDB is incapable of.

Our contribution presented in this paper is a flexible and modular framework model of an encrypted database that is capable of executing a wide range of queries including those containing complex arithmetic expressions over encrypted data in the database, multiple predicates, heavy relational operations like joins, grouping, and more. Also we briefly discuss approaches to extend this model in order to improve its security properties, add more data types, and extend the range of supported queries.

This work is organized as follows: Section II introduces the threat and security model that this work is based upon. Section III fully describes the proposed model, all its parts, and their interactions. Section IV describes the research directions to extend the proposed model. Section V summarizes the results of the work reflected in this paper.

II. Security and Threat Model

In order to discuss a secure, privacy-preserving model, we first need to establish what are the threats we are considering and what does it mean that a system is *secure* against these threats.

The adversary is modeled as a passively curious agent who has *full* access to the cloud infrastructure that hosts the DBMS. He never actively interferes with the work- and data-flow in the system, limiting himself to passive observation. All communications are assumed to be secure. Proxy (see Section III-B) is assumed to be fully trusted, but in Section IV we will lower that requirement and will only require proxy to be semi-honest, i.e., it follows the instructions and protocols but is maliciously curious. Effectively, that lowers the required trust level of the proxy to that of the database server.

Proposed model is supposed not to let an adversary to obtain plaintext data in a reasonable time with non-negligible probability (i.e., in a way other than guessing the decryption keys).

III. Proposed Model

The model discussed in this section only supports numeric data. Although it could easily be extended to support textual or other types of data, we will not yet do that in order not to over-encumber the model, since predicates on numeric data are generally more complex than those on textual data. E.g., queries on textual data are most likely to be simple search queries like `SELECT * FROM t WHERE t.a = "apple"`, and we are more concerned about executing queries that contain multiple different operations in the predicate like `... WHERE t.a + t.b * t.c < t.d`. We will however consider this extension in Section IV-A.

It is important to note that this work does not propose a specific solution to the problem of an encrypted DBMS but rather a framework model upon which such a solution could be built.

Apart from introducing to the model the support for textual data, in Section IV we will also discuss some other significant directions, in which it could be extended.

A. Cryptographic Abstractions

In this work we do not consider any specific cryptographic schemes. Instead, we consider two cryptographic abstractions: an encryption and a hash.

1) **Encryption.** The encryption abstraction is a pair of functions that we will denote $e(x, k_e)$ and $d(x, k_d)$, where $d(e(x, k_e), k_d) = x$, k_e and k_d are the encryption and decryption keys respectively, and it is possible that

$k_e \neq k_d$, which reflects the symmetric and asymmetric encryption schemes.

2) **Hash.** The hash abstraction is a single function, that we will denote $h(x, k)$, where $\nexists g : g(h(x, k), k) = x$, and k is a secret key.

B. Model Architecture

The basic version of the proposed model, which will be extended in Section IV consists of three entities — a DBMS, a user, and a proxy (see Fig. 1).

DBMS: The DBMS is running in a completely untrusted (as defined by the threat model) environment of a cloud platform provider. Thus it can never access the plaintext data or en-/decryption keys.

User: The user is completely trusted and is generally supposed to possess a very limited computational resources (e.g., accessing the system from a thin client or a mobile device).

Proxy: The proxy is an intermediary agent between the user and the DBMS and it possesses a *sufficient* amount of computational resources. In a simplified model proxy is fully trusted, but in Section IV we will discuss ways to lower the required trust level for the proxy or proxies.

C. Storage Model

We store a special entity called a *cryptoset* in the encrypted database for every plaintext data value. Let v be the plaintext value, then the cryptoset that the encrypted database stores is $\{e_1(v, k_1^e), \ldots, e_n(v, k_n^e), h_1(v, k_1^h), \ldots, (v_m, k_m^h), u\}$, where $e_i, 1 \leq i \leq n$ are encryption functions; $h_i, 1 \leq i \leq m$ are cryptographic hashing functions; and u is a flag denoting whether the cryptoset is in a *consistent* state or not. k_i^* is an encryption key corresponding to a specific encryption or a hashing function.

Encryption Functions: These functions produce the encryptions by different encryption schemes that facilitate execution of various operations. An addition-homomorphic encryption scheme like Paillier or BGN[1] could be used to perform encrypted addition, multiplication-homomorphic scheme like ElGamal — to perform encrypted multiplication.

It could also be a deterministic encryption to perform search, or an order-preserving encryption to perform greater/less checks, or any other encryption scheme that is suitable for required purposes.

Hashing Functions: Some implementations of search, greater/less checks, and possibly some other operations require not only the encryption of the plaintext value but also some additional data in the form of a cryptographic hash [10]. So the cryptoset is designed to store that data too.

[1]Boneh-Goh-Nissim

Consistency Flag: As long as the cryptoset stores a *set* of values that are derived in various ways from the original plaintext value, certain modifications to the cryptoset could possibly bring it to the inconsistent state, i.e., if $\exists i, j : 1 \leq i, j \leq m + n,\ v_i \neq v_j$ for the cryptoset $\{e_1(v_1, k_1^e), \ldots, e_n(v_n, k_n^e), h_1(v_{n+1}, k_1^h), \ldots, (v_{n+m}, k_m^h), u\}$, then this cryptoset is in an inconsistent state.

Initially, when created, all cryptosets are in a consistent state and the flag u is set to 0. If a certain operation executed by the database (e.g., an `UPDATE` query that performed a homomorphic operation and substituted one of the encryptions in the cryptoset by the result of the operation) amends a cryptoset, then it sets the flag u to a non-zero value that allows the database to determine which encryption in the cryptoset carries the up-to-date value. DBMS then proceeds to perform a re-encryption of the cryptoset, which is considered in the next section. Until a re-encryption is complete, marked cells of the table are considered locked for both reading and writing by all other queries.

D. Maintaining Cryptoset Consistency

After a user has issued a query that resulted in an amendment of certain cryptosets in the database, and DBMS has marked them as inconsistent using the consistency flags, it sends cryptosets that are pending re-encryption to the proxy, as illustrated by the Algorithm 1.

Generally, if the DBMS supports a simple limited subset of SQL, namely `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, it should only do that after an `UPDATE` query: `SELECT` does not change the data in the database; `DELETE` removes certain cryptosets completely, so we don't need to re-encrypt them; `INSERT` creates new cryptosets, which are by definition in a consistent state.

Algorithm 1 Consistency maintaining by DBMS

```
if query could have modified data then
  for all tables t that were affected do
    for all columns c that were affected do
      T ← SELECT c FROM t WHERE t.c.u ≠ 0
      send set T to the proxy
    end for
  end for
end if
```

As soon as the proxy receives a set of cryptosets to re-encrypt, it proceeds to execute Algorithm 2. First, it inquires (algorithm INQ) of the user the decryption key for the amended encryption (determined by u) and the encryption keys for all other encryptions and keys for hashes. Proxy then proceeds to obtain a plaintext of an updated value by decrypting the amended encryption, after which it encrypts the obtained plaintext with other needed encryptions using the encryption key supplied by the user. It then sets u to zero and sends re-encrypted cryptosets to the server in order

Algorithm 2 Consistency maintaining by proxy

Require: set T received from DBMS
  for all cryptosets c in T **do**
    $(k_u^d, (k_1^e, ..., k_{u-1}^e, k_{u+1}^e, ..., k_n^e, k_1^h, ..., k_m^h)) \leftarrow$ INQ
    $v \leftarrow d(c.e_u, k_u^d)$
    $res \leftarrow ()$ {an empty set}
    for $i = 1$ to n **do**
      if $i \neq u$ **then**
        PUSH$(res, e_i(v, k_i^e))$ {pushes value into the set}
      else
        PUSH$(res, c.e_u)$
      end if
    end for
    for $i = 1$ to m **do**
      PUSH$(res, h_i(v, k_i^h))$
    end for
    DELETE $(k_u^d, (k_1^e, ..., k_{u-1}^e, k_{u+1}^e, ..., k_n^e, k_1^h, ..., k_m^h))$
    PUSH$(res, 0)$ {zeroed consistency flag}
    send res to DBMS
  end for

to replace the inconsistent ones. When they are no longer needed, proxy deletes all the keys from its memory.

E. Query Execution Workflow

Suggested model is based on the following assumption:

Let O be a set of operations that are required to be supported by the DBMS: greater/less/equality checks, addition, multiplication, *etc*. For every plaintext operation $f \in O$, where f is in a form of a function $f(x_1, \ldots, x_n)$, $n \geq 1$, there exist encryption schemes $e_1^f, \ldots, e_m^f$, hashing functions $h_1^f, \ldots, h_k^f$, a decryption function d, and a function g_f such that $d(g_f(e_1^f(x_1), \ldots, e_m^f(x_1), h_1^f(x_1), \ldots, h_k^f(x_1), \ldots, e_1^f(x_n), \ldots, e_m^f(x_n), h_1^f(x_n), \ldots, h_k^f(x_n))) = f(x_1, \ldots, x_n)$.

Query is executed generally in 4 steps.

1) **Query issuance.** User generates a plaintext query, couples it with *corresponding* encryption keys, and transfers it all to the proxy. *Corresponding* means those that correspond to the encryption or hashing functions needed to perform an encrypted counterpart of plaintext operations that are present in the issued query predicate.
2) **Query modification.** Upon receiving the query and the keys from the user, proxy performs a query modification in 3 stages:
   a) For every plaintext operation f that is present in the query, it is substituted by its encryption-oriented counterpart g_f.
   b) Every reference to an attribute is narrowed down to a reference to a specific element of the corresponding cryptoset.
   c) Every plaintext constant in the query is encrypted or hashed with the corresponding scheme.

   Example: Let `DET()` be a deterministic encryption, `OPE()` — an order-preserving encryption. If the original plaintext query was `SELECT * FROM t WHERE t.x = 7 AND t.y > 10` then these 3 stages will be:

   a) `t.x = 7` $\rightarrow g_=$(`t.x`, 7)
      `t.y > 10` $\rightarrow g_>$(`t.y`, 10)
   b) $g_=$(`t.x`, 7) $\rightarrow g_=$(`t.x.DET`, 7)
      $g_>$(`t.y`, 10) $\rightarrow g_>$(`t.y.OPE`, 10)
   c) $g_=$(`t.x.DET`, 7) $\rightarrow g_=$(`t.x.DET`,`DET`(7))
      $g_>$(`t.y.OPE`, 10) $\rightarrow g_>$(`t.y.OPE`,`OPE`(10))

   Eventually, the query is going to look like this: `SELECT * FROM t WHERE` $g_=$(`t.x.DET`, `DET`(7)) `AND` $g_>$(`t.y.OPE`, `OPE`(10)).
   Proxy then transfers the query to the DBMS.
3) **Query execution.** As long as DBMS knows how to execute the operations g_*, it can just proceed to executing the query. The tricky part here is if a query involves nested operations. Let us consider a query `SELECT * FROM t WHERE t.a + t.b * t.c > t.d`. The predicate `t.a + t.b * t.c > t.d` could be rewritten as follows $f_>(f_+($`t.a`, $f_\times($`t.b`,`t.c`$)$),`t.d`$)$, which would be transformed by a proxy into $g_>(g_+($`t.a.ADD`, $g_\times($`t.b.MUL`,`t.c.MUL`$)$`.ADD`$)$`.OPE`,`t.d.OPE`$)$, where `MUL` and `ADD` are multiplication- and addition-homomorphic encryption schemes. The problem here is that, e.g., the result of $g_\times($`t.b.MUL`,`t.c.MUL`$)$ is a ciphertext c encrypted by `MUL` and DBMS is unable to produce an addition-homomorphic ciphertext c`.ADD`. Thus, DBMS should store such intermediate results of computations in temporary columns and invoke an algorithm of consistency maintaining by proxy (Algorithm 2) to produce required encryptions from the encryptions that were obtained as a result of operation. After required encryptions were received from the proxy, the DBMS continues the execution of the query.
   If the query is expecting results from the DBMS (e.g., a `SELECT` query), then the DBMS sends the resulting set back to the proxy, and the system proceeds to the next step.
4) **Results decryption (*optional*).** Upon receiving the query results, the proxy selects one encryption in the cryptoset (e.g., the one whose decryption function if fastest to compute), and inquires of the user the decryption keys to that specific encryption. Upon receiving, it proceeds to decrypt all the results and then sends plaintext results to the user.
   This stage is optional since not all the queries produce a result set, e.g., `DELETE` queries do not.

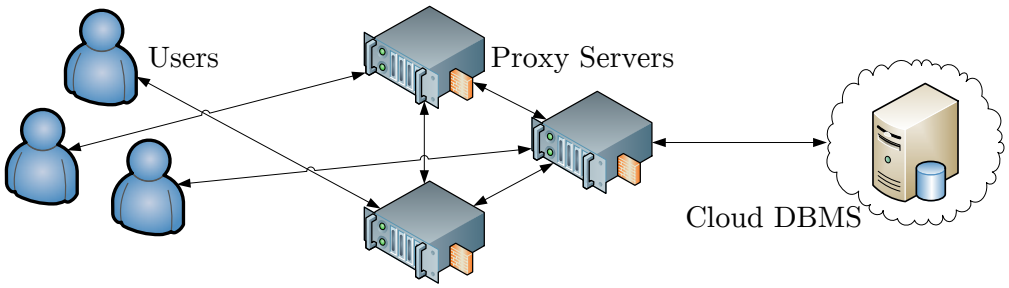

Figure 2: Extended model architecture

F. Multi-user Setting

Existing approaches to locking data and scheduling queries are well-applicable to the discussed model. In order for them to work we just need to ensure that while the cryptoset is in an inconsistent state, it is fully locked for both reading and writing, i.e., ensure atomic update of cryptoset.

IV. Model Extensions

Some of the extensions to the proposed model that we are going to discuss in this section are

A. Data types other than numeric
B. Complex expressions in other SQL clauses than the `WHERE` clause
C. Lower trust proxies

A. Non-numeric Data Types

Until now the only data types supported by the proposed model were numeric. A typical cryptoset for numeric value v would look somewhat like $(\texttt{ADD}(v), \texttt{MUL}(v), \texttt{DET}(v), \texttt{OPE}(v))$. Another widely used data type in databases is text. Textual data is even easier to work with as long as it usually is not involved in complex nested operations and often is just used in queries in the context of a simple exact matching: `SELECT * FROM t WHERE t.name = "Alice"`. Different data type would just require different set of encryptions in the cryptoset, e.g., for textual data cryptoset could consist of just one deterministic encryption.

It is worth noting that encrypted search (for any data type) implemented using deterministic encryptions is the most basic approach and there are various disadvantages to it. We use it here just for the illustrative purposes. One of the approaches to the encrypted search that is lacking many of said disadvantages and is suitable to be incorporated in the model proposed in this paper is described in one of our earlier works [10].

B. Performing Operations in Other Clauses

The designed workflow of computing the expressions does not in any way rely on this expressions being in a predicate and thus can be extended to be used in case an expression appears somewhere else in the query, e.g., `SELECT (t.a + t.b) * t.c FROM t`.

C. Proxies with Lower Trust Level

In the proposed model we assume the proxy to be fully trusted as we are required to share the secret en-/decryption keys with it. However, in our previous works we suggested an approach that allows distribute a secret among several parties in a way that none of them knows the whole secret but still they can en-/de-/re-encrypt data if working together [10], [11]. That will require certain changes in the architecture of the model, as shown in Fig. 2. The suggested approach was only investigated for the specific encryption scheme (ElGamal), and a research needs to be done in order to select and probably adjust other encryption schemes that could be deployed into a system in a similar way.

Such an approach makes it possible to lower the required trust level for the proxy and thus enhances the overall system security. On the other hand, it requires several proxies instead of just one, moreover, these proxies need to be physically and semantically[2] far from each other in order to lower the chance of collusion, which may increase the total cost of having a practical implementation of the proposed model.

V. Conclusion

This paper presents a flexible and extensible abstract model of an encrypted database management system, which is able to execute queries that contain arbitrary combinations of basic operations over encrypted data, e.g., homomorphic addition and multiplication, greater/less/equal checks, search, etc. Being abstract, this model does not incorporate any specific implementations of the basic operations over encrypted data, and the fact that implementations of various operations do not affect each other under this model makes it extremely flexible, allowing a potential user to have only operations he needs and thus reducing the complexity and overhead of the resulting practical system.

[2]E.g., administered by different people.

REFERENCES

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[2] C. Gentry and S. Halevi, "Implementing gentry's fully-homomorphic encryption scheme," in *Advances in Cryptology – EUROCRYPT 2011*, ser. Lecture Notes in Computer Science, K. Paterson, Ed. Springer Berlin Heidelberg, 2011, vol. 6632, pp. 129–148. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-20465-4_9

[3] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," 2012.

[4] H. Hacıgümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-service-provider Model," in *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '02. New York, NY, USA: ACM, 2002, pp. 216–227. [Online]. Available: http://doi.acm.org/10.1145/564691.564717

[5] R. Popa, N. Zeldovich, and H. Balakrishnan, "CryptDB: A Practical Encrypted Relational DBMS," Technical Report MIT-CSAIL-TR-2011-005, MIT, Tech. Rep., 2011.

[6] R. Popa, C. 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*. ACM, 2011, pp. 85–100.

[7] ——, "CryptDB: Processing Queries on an Encrypted Database," *Communications of the ACM*, vol. 55, no. 9, pp. 103–111, 2012.

[8] Z. Yang, S. Zhong, and R. Wright, "Privacy-preserving queries on encrypted data," *Computer Security–ESORICS 2006*, pp. 479–495, 2006.

[9] B. Hore, S. Mehrotra, and G. Tsudik, "A privacy-preserving index for range queries," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, pp. 720–731. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316752

[10] V. Sidorov and W. K. Ng, "Complex queries in a shared multi user relational cloud database," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 903–909. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2013.49

[11] T. H. Nguyen, H. G. Do, W. K. Ng, and H. Zhu, "Cloud-enabled data sharing model," in *Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on*, June 2012, pp. 1–6.