Static Analysis and Program Transformation for Secure Computation on the Cloud

Yao Dong
Rensselaer Polytechnic Institute
Troy, New York
dongy6@rpi.edu

Ana Milanova
Rensselaer Polytechnic Institute
Troy, New York
milanova@cs.rpi.edu

Julian Dolby
IBM Thomas J. Watson Research Center
Yorktown Heights, New York
dolby@us.ibm.com

ABSTRACT

Cloud computing allows clients to upload data and computation to untrusted servers, which leads to potential violations to the confidentiality of client data. We propose a static program analysis which transforms a Java program into an equivalent one, so that it performs computation over encrypted data and preserves data confidentiality.

The proposed analysis consists of two stages. The first stage is a type-based information flow analysis which partitions the program so that only sensitive part needs to be encrypted. The second stage is an inter-procedural data flow analysis which deduces the appropriate encryption scheme for sensitive variables. In the end, we show the preliminary experimental results and outlines the research directions.

1. PROBLEM

With the booming of Internet-based business, the number of applications running over the Cloud has increased dramatically over the past few years. These applications have created a new challenge. Cloud-based applications usually outsource the computation and data storage to third parties such as Amazon EC2 and S3 which can lead to violations of the confidentiality and integrity of sensitive data. The problem is, can programs running on the cloud perform computation over encrypted data?

2. APPROACH

One approach to address this problem is to make use of fully homomorphic encryption [7]. This cryptographic scheme makes it possible to compute arbitrary functions over encrypted data on the server. Unfortunately, current implementations of fully homomorphic encryption schemes are prohibitively expensive by orders of magnitude [3, 8, 9].

Another approach is to encrypt different data using different encryption schemes. Some specialized encryption schemes are more efficient for specific computations over encrypted data. Therefore, if some data is only involved in certain operations, it can be encrypted using an appropriate encryption scheme that supports those operations, as in CryptDB [23] and MrCrypt [27]. Shah et al. [26] conjectured that program partitioning [2, 30] and other program analysis techniques can help (1) minimize computation on the untrusted server, (2) deduce efficient encryption schemes (for data that is involved only in operations supported by a given scheme) and (3) deduce re-encryption points (for data that is involved in multiple operations, not all supported by any given scheme).

We propose novel program analysis techniques to address this problem. Specifically, we propose a system that *automatically* analyzes and transforms a Java program into an equivalent one that performs secure computations over encrypted data on an untrusted server. Given a Java program in which the user has marked certain variables as *sensitive*, our system analyzes and determines the appropriate encryption schemes for the sensitive variables based on the operations performed on those variables. We choose more efficient schemes instead of fully homomorphic encryption to encrypt sensitive data.

2.1 Information Flow Analysis

Given a program, our system first partitions the variables in the program into two parts: sensitive variables, which must be encrypted, and cleartext ones which may remain in cleartext. The key insight is that if there is information flow from the user-provided sensitive data (i.e., the *sources*) to a variable x, then x must be encrypted. Conversely, if there is no flow from sensitive data to a variable, then the variable and its operations may remain in cleartext form. Our analysis maximizes the part that remains in cleartext. Further, our analysis is context-sensitive, which allows for the same method to be applied on encrypted values and on cleartext values.

Consider the program in Fig. 1. Only variable s in m is marked sensitive by the user, meaning that it contains sensitive information and must be manipulated in encrypted form. Since there is information flow from s to field d and then to ss, variable ss becomes sensitive. Variable c is clear and cc is clear as well. Note that the field d and the methods get and set are polymorphic, as they operates over a sensitive argument at line 16, and over a clear argument at line 20.

The information flow analysis consists of a type system and a corresponding type inference. We built them upon our framework for inference and checking of pluggable types [10, 20], which we have used to define and implement many practical type-based analyses [10, 13, 11, 12].

The type system has three qualifiers: sensitive, clear, and poly, which are used to indicate which variables are sensitive,

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

```
1   public class Data {
2     poly int d;
3     poly int get(poly Data this) {
4       if (this.d < 0)
5         this.d = this.d + 1;
6       return this.d;
7     }
8     void set(poly Data this, poly int p) {
9       this.d = p;
10    }
11  }
12  public class Example {
13    public void m() {
14      sensitive Data ds = new Data();
15      sensitive int s = ...; // sensitive soure
16      ds.set(s);
17      sensitive int ss = ds.get();
18      clear Data dc = new Data();
19      clear int c = ...;
20      dc.set(c);
21      clear int cc = dc.get();
22      ...
23    }
24  }
```

Figure 1: A sample Java program.

```
1   public class Data {
2     int d;
3     int d_Sen;
4     int get_Sen(Data this) {
5       if (d_Sen < 0)
6         this.d_Sen = this.d_Sen + 1;
7       return this.d_Sen;
8     }
9     void set_Sen(int p) { this.d_Sen = p; }
10    int get(Data this) {
11      if (this.d < 0) this.d = this.d + 1;
12      return this.d;
13    }
14    void set(int p) { this.d = p; }
15  }
16  public class Example {
17    public void m() {
18      Data ds = new Data();
19      int s = ...;
20      ds.set_Sen(s);
21      int ss = ds.get_Sen();
22      Data dc = new Data();
23      int c = ...;
24      dc.set(c);
25      int cc = dc.get();
26      ...
27    }
28  }
```

Figure 2: The transformed program. The methods get_Sen and set_Sen are the sensitive versions of get and set.

clear or polymorphic. There is a subtyping relation between the qualifiers clear <: poly <: sensitive which forbids an assignment from a sensitive variable to a poly or clear one.

Given a set of sensitive sources, type inference *automatically* types each variables in the program according to the constraints created for all program statements. The inferred types for the program in Fig. 1 are shown in front of each variables. Class Data is inferred as polymorphic, ds and ss are inferred as sensitive. Finally dc, c and cc, remain clear since there is no flow from source s to any of these.

After the information flow analysis, every variable has a type sensitive, poly or clear. All methods that have poly parameters (include implicit parameter this) will have two versions, one for computation over encrypted data and the other for computation over cleartext data. The transformed program is show in Fig. 2. All sensitive and poly variables are given an encryption type which is discussed in the next section.

2.2 Data Flow Analysis

As in CryptDB [23], we use four efficient encryption schemes to support certain operations. They are (1) RND: a randomized encryption scheme for variables not used in any operations; (2) AH: an additively homomorphic encryption scheme for variables only used in addition operations; (3) DET: a deterministic encryption scheme for variables involved in equality checking; and (4) OPE: an order-preserving encryption scheme for variables used in comparison operations.

As not all operations will be supported by any given encryption scheme, it may not be enough to give just one encryption type for some variables. For example, at line 4 of Fig. 1, field d needs OPE type, while at line 5, it needs AH type. No matter which type d is, we must decrypt the ciphertext and re-encrypt it using the other type. We call this a *type conversion* or *re-encryption*. Our goal is to *minimize and even completely eliminate type conversions* because the re-encryption process is expensive.

An intuitive idea to address the problem is that we can encrypt the value of d by two encryption types (AH and OPE) on the secure client end and then send the two versions of ciphertext to the untrusted server. The program will choose the OPE version at line 4 and the AH version at line 5 *without any type conversions*.

Based on this idea, we propose a context-insensitive interprocedural data-flow analysis inspired by Available Expressions. Initially, each variable x is initialized to all four types $S(\mathsf{x}) = \{\text{RND, AH, DET, OPE}\}$. The transfer functions depend on specific statements, which are described as follows:

- x = y: $S(\mathsf{x}) = S(\mathsf{y})$;
- x = y.f: $S(\mathsf{x}) = S(\mathsf{f})$;
- x.f = y: $S(\mathsf{f}) = S(\mathsf{f}) \cap S(\mathsf{y})$;
- x = y.m(z): $S(\mathsf{x}) = S(\mathsf{ret_m})$, $S(\mathsf{p}) = S(\mathsf{p}) \cap S(\mathsf{z})$;
- x = y + z: $S(\mathsf{x}) = \{\text{AH}\}$.

$\mathsf{ret_m}$ and p are the return value and parameter of m. The analysis is field-based, that is, field f is considered as a global variable. All other components of the data-flow analysis are the same as a standard Available Expressions, such as, the

direction of the data flow is Forward, and the meet operator is Must.

The analysis examines the operands of arithmetic operations. If the corresponding encryption type is available then there is no need for conversion. If it is not available, then we cannot avoid the type conversion. For example, OPE is available at line 5 of Fig. 2 since initially $S(\mathsf{d\_Sen}) = \{\text{RND, AH, DET, OPE}\}$ and no operation expressions change it. Similarly, AH is also available at line 6. Therefore, the program can execute over encrypted data without any type conversions. It is worth noticing that the addition expression at line 6 redefines d_Sen so that $S(\mathsf{d\_Sen}) = \{\text{AH}\}$. If there is any operations of d_Sen other than addition afterwards, then type conversions there will be unavoidable.

We built the data-flow analysis on top of a general-purpose inter-procedural analysis framework for Soot [22]. This framework uses data flow values for context-sensitivity which is a good fit for our analysis. We use a mapping of variables to a set of encryption types as data flow values in our analysis. In addition, the framework provides program representations for Soot's Jimple IR which is compatible with our checker framework [20] used in the first stage analysis.

In preliminary results, we have applied our analysis to 10 benchmarks, which are general-purpose Java programs. 7 come from the Java Olden (JOlden) benchmark suite [1], which implement different algorithms using the tree data structure. The other 3 benchmarks are medium-sized ones from the EnerJ benchmark suite [25].

Of the 10 programs, our analysis determines that 6 require type conversions. A typical example of type conversion is shown in Fig. 3. Here rating comes from an array of sensitive data. The addition operation makes $S(\mathsf{sumRatings}) = \{\text{AH}\}$. The type for division [2] is not available at sumRatings / totalReviews; hence, a type conversion from AH to the division type is necessary at line 6. Similarly, another type conversion from AH to OPE is also required at line 9 for variable fraction.

```
1  ...
2  while ( ... ) {
3    sumRatings += rating; // rating is sensitive
4    totalReviews++;
5  }
6  avgReview = sumRatings / totalReviews;
7  absReview = Math.floor(avgReview);
8  fraction = avgReview − absReview;
9  if (fraction < (division ∗ i)) {
10   outValue = absReview + division ∗ i;
11 }
12 ...
```

Figure 3: A slice of program requiring type conversions.

3. CHALLENGES

We outline several challenges.

- Can we further reduce and finally completely avoid type conversions (re-encryption)? There might be two solutions to achieve this goal. One possible solution is to refactor the program so that there is no need for type conversions. The other is to move a slice of the program to the trusted client so that the computation can be performed over cleartext securely, and then send the encrypted result back to the server which will continue with the rest of program. The problem is how to minimize this slice.

  Take Fig. 3 as an example. We can extract lines 6 to 11 into a method m_client which will run on the client where the computation can be done over cleartext values. Then the client sends the encrypted result into outValue back to server (only outValue will be used in the rest of program running on the server). The expected transformed program on the server will look as follows:

  ```
  while ( ... ) {
    sumRatings += rating; // rating is sensitive
    totalReviews++;
  }
  outValue = m_client(sumRatings, totalReviews);
  ```

- In order to better evaluate our approach, we would like to find and analyze more benchmarks. We conjecture that a large percentage of Java programs would be "separable", i.e., would not need type conversions. For cases where type conversions cannot be avoided, we will investigate the two approaches we outlined in the previous bullet and investigate programming effort and performance degradation. In the future we plan to analyze Hadoop MapReduce programs since they are popular and widely used for cloud computing [17, 4, 28, 16, 6, 15, 14, 31, 18, 21, 24, 29, 19, 1, 5].

- Currently, our system does not analyze Java libraries. However, certain library methods may perform operations on the variables passed as arguments. For example, hash-based containers imply equality checking for the container elements. java.lang.Math class includes a large variety of operations. We must handle the Java library to make our system practical. Our prior work with incomplete programs (e.g., libraries [13], Android apps [12]) will be a good starting point.

- Besides data security, program performance is also very important. We would like to actually run the transformed programs over encrypted data in order to verify that our approach with efficient encryption schemes has better performance than fully homomorphic encryption schemes. The challenge here is that it is difficult to find suitable Java programs containing only integer and String variables, which is what we need because existing implementations of encryption schemes are limited to integer values.

4. RELATED WORK

[1] https://github.com/farseerfc/purano/tree/master/src/jolden

[2] We do not specify an encryption type for division operation in order to simplify our description. It is very easy to extend our system to handle any other operation just like addition as AH. For example, we can simply add a DIV type for division operaion to the initial set and a transfer function $S(\mathsf{x}) = \{\text{DIV}\}$ for statement x = y / z.

MrCrypt [27] is a system that provides secure computations over encrypted data on the cloud server. The key difference between these two systems is that our system performs analysis on arbitrary Java program while MrCrypt handles only a small functional subset of Java, such as MapReduce programs which require only a small set of operations. Furthermore, out system use information flow analysis and parametric polymorphism to minimize the computation of encryption, while MrCrypt has to encrypt the entire program which results in a relatively larger encryption overhead. Finally, out system handles programs where more than one operations apply on the same variable, while MrCrypt fails on such programs. CryptDB [23] is another system that can peform computation over encrypted data. It analyzes SQL queries and rewrites them to execute on SQL databases. Like MrCrypt, CryptDB is also limited to a specific application — database queries.

Program partitioning techniques have been used to provide efficient computation in other domains. EnerJ [25] is a type-based system that partitions a program into a precise component and an approximate component. Computing on approximate data is less expensive than computing on precise data. Therefore running the program can save energy at some accuracy cost. Similarly to EnerJ, our system partitions the program based on information flow. One key difference is that EnerJ requires significant amount of programmer-provided annotations, while our system automatically infers a partition with a maximal number of cleartext variables.

Another work related to program partitioning is Swift [2], an approach to partition a program into two parts, one running on the client and the other running on the server. The difference between our system and Swift is that Swift physically isolates security-critical code from the untrusted environment to provide security, while our system marks sensitive code and encrypts the data so that the whole program can run on the untrusted server. Further, Swift requires programmers to use a security-typed programming language, Jif/split [30] to write programs. This increases the burden on programmers and it may not be suitable for existing programs. In contrast, our system only requires a few annotations by the programmer to indicate sensitive sources.

5. REFERENCES

[1] N. L. S. H. A. Haider, X. Yang and X.-H. Sun. Ic-data: Improving compressed data processing in hadoop. In *in Proc. of 22nd annual IEEE International Conference on High Performance Computing (HiPC 2015)*, 2015.

[2] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 31–44, New York, NY, USA, 2007. ACM.

[3] M. Cooney. Ibm touts encryption innovation: New technology performs calculations on encrypted data without decrypting it. *Network World*, June 2009.

[4] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross. Codes: Enabling co-design of multilayer exascale storage architectures. In *Proceedings of the Workshop on Emerging Supercomputing Technologies 2011*, 2011.

[5] B. Feng, N. Liu, S. He, and X.-H. Sun. Hpis3: towards a high-performance simulator for hybrid parallel i/o and storage systems. In *Proceedings of the 9th Parallel Data Storage Workshop*, pages 37–42. IEEE Press, 2014.

[6] J. Fu, N. Liu, O. Sahni, K. E. Jansen, M. S. Shephard, and C. D. Carothers. Scalable parallel i/o alternatives for massively parallel partitioned solver systems. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[7] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[8] C. Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, Mar. 2010.

[9] C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual International Conference on Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT'11, pages 129–148, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] W. Huang, W. Dietl, A. Milanova, and M. D. Ernst. Inference and checking of object ownership. In *Proceedings of the 26th European Conference on Object-Oriented Programming*, ECOOP'12, pages 181–206, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] W. Huang, Y. Dong, and A. Milanova. Type-based taint analysis for java web applications. In *Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering - Volume 8411*, pages 140–154, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[12] W. Huang, Y. Dong, A. Milanova, and J. Dolby. Scalable and precise taint analysis for android. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 106–117, New York, NY, USA, 2015. ACM.

[13] W. Huang, A. Milanova, W. Dietl, and M. D. Ernst. Reim & reiminfer: Checking and inference of reference immutability and method purity. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pages 879–896, New York, NY, USA, 2012. ACM.

[14] N. Liu, C. Carothers, J. Cope, P. Carns, and R. Ross. Model and simulation of exascale communication networks. *Journal of Simulation*, 6(4):227–236, 2012.

[15] N. Liu, C. Carothers, J. Cope, P. Carns, R. Ross, A. Crume, and C. Maltzahn. Modeling a leadership-scale storage system. *Parallel Processing and Applied Mathematics*, pages 10–19, 2012.

[16] N. Liu and C. D. Carothers. Modeling billion-node torus networks using massively parallel discrete-event simulation. In *Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on*, pages 1–8. IEEE, 2011.

[17] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. On the role of burst buffers in leadership-class storage systems. In *012 IEEE 28th Symposium on Mass Storage Systems*

and Technologies (MSST), pages 1–11. IEEE, 2012.

[18] N. Liu, A. Haider, X.-H. Sun, and D. Jin. Fattreesim: Modeling large-scale fat-tree networks for hpc systems and data centers using parallel and discrete event simulation. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 199–210. ACM, 2015.

[19] N. Liu, X. Yang, X.-H. Sun, J. Jenkins, and R. Ross. Yarnsim: Simulating hadoop yarn. 2015.

[20] A. Milanova and W. Huang. Inference and checking of context-sensitive pluggable types. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 26:1–26:4, New York, NY, USA, 2012. ACM.

[21] L. NING, F. JING, D. C. CHRISTOPHER, O. SAHNI, E. J. KENNETH, and M. S. SHEPHARD. Massively parallel i/o for partitioned solver systems. *Parallel Processing Letters*, 20(04):377–395, 2010.

[22] R. Padhye and U. P. Khedker. Interprocedural data flow analysis in soot using value contexts. In *Proceedings of the 2Nd ACM SIGPLAN International Workshop on State Of the Art in Java Program Analysis*, SOAP '13, pages 31–36, New York, NY, USA, 2013. ACM.

[23] 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*, SOSP '11, pages 85–100, New York, NY, USA, 2011. ACM.

[24] M. Rasquin, P. Marion, V. Vishwanath, B. Matthews, M. Hereld, K. Jansen, R. Loy, A. Bauer, M. Zhou, O. Sahni, et al. Electronic poster: co-visualization of full data and in situ data extracts from unstructured grid cfd at 160k cores. In *Proceedings of the 2011 companion on High Performance Computing Networking, Storage and Analysis Companion*, pages 103–104. ACM, 2011.

[25] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

[26] M. Shah, E. Stark, R. A. Popa, and N. Zeldovich. Language support for efficient computation over encrypted data, January 2012.

[27] S. D. Tetali, M. Lesani, R. Majumdar, and T. Millstein. Mrcrypt: Static analysis for secure cloud computations. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 271–286, New York, NY, USA, 2013. ACM.

[28] K. Wang, N. Liu, I. Sadooghi, X. Yang, X. Zhou, T. Li, M. Lang, X.-H. Sun, and I. Raicu. Overcoming hadoop scaling limitations through distributed task execution. In *2015 IEEE International Conference on Cluster Computing*, pages 236–245. IEEE, 2015.

[29] X. Yang, N. Liu, B. Feng, X.-H. Sun, and S. Zhou. Porthadoop: Support direct hpc data processing in hadoop. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 223–232. IEEE, 2015.

[30] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers. Untrusted hosts and confidentiality: Secure program partitioning. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 1–14, New York, NY, USA, 2001. ACM.

[31] D. Zhao, N. Liu, D. Kimpe, R. Ross, X.-H. Sun, and I. Raicu. Towards exploring data-intensive scientific applications at extreme scales through systems and simulations. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1824–1837, 2016.