Privacy-Preserving Data Exfiltration Monitoring Using Homomorphic Encryption

Kurt Rohloff
Department of Computer Science
New Jersey Institute of Technology
Newark, New Jersey 07102
Email: rohloff@njit.edu

***Abstract*—Monitoring and encryption are essential to secure today's computer networks. Monitoring network traffic data can be especially useful to protect against data exfiltration by detecting signatures in file metadata to identify especially sensitive files that should not be publicly released. Encryption is also a useful data protection tool by reducing access to file to users with the correct decryption keys. However, encryption restricts the visibility of signatures, but this may be needed because some signatures used to protect against data exfiltration may themselves be sensitive, as knowledge of signatures could help adversaries circumvent monitoring. We present the design and experimental results on a practical exfiltration guard to securely and privately monitor flows of encrypted information for encrypted signatures without requiring the decryption of the data flows or the signatures or the sharing of decryption keys. The basis from our approach comes from recent advances in the design and implementation of homomorphic encryption which enables secure computing on encrypted data. We show experimental results from applying our exfiltration monitoring protocol with a prototype proof-of-concept encrypted data guard running on a commodity computing hardware. These designs point to possible future advances driven by ongoing homomorphic encryption improvements to compute on encrypted data for more advanced and secure filtering and exfiltration protection schemes.**

I. Introduction

Monitoring and encryption are essential to secure today's computer networks. As identified in analyses of recent cyber-attacks, broader application of practical exfiltration monitoring technologies could aid to better protect sensitive information [1] from being stolen from sensitive data stores. For example, the perpetrators of the APT1 attacks exfiltrated vast amounts of sensitive information and there were few safe-guards in place to prevent this exfiltration once adversaries obtained inside control of their target networks. Strong encryption technologies could partially address data exfiltration concerns by preventing adversaries from being able to use any exfiltrated encrypted information. However, strong encryption is not a panacea and encryption could be used as a cover to prevent the practical monitoring of information flows [2]. Monitoring requires visibility into data, while encryption restricts visibility.

We present an approach that enables data to be encrypted using a secure post-quantum scheme, but allows data leaving a network to be monitored for the presence of signatures which indicate that data is approved to be exported (or not.) Until now there has been relatively limited ability to monitor and identify the exfiltration of encrypted data, particularly with respect to situations when the signatures themselves might be sensitive, as knowledge of signatures could help adversaries circumvent monitoring. Sensitive signatures are difficult to use securely without risk of leakage on host-based systems. Recent advances in functional encryption [3] have suggested approaches to evaluate the presence of signatures in encrypted data, resulting in a decrypted output that determines whether the evaluated encrypted data is safe to release. However, there has been limited progress in implementing usable functional encryption.

We apply recent implementation results that demonstrate practical Somewhat Homomorphic Encryption (SHE) [4], [5], [6]. Such an exfiltration data guard would be useful to support Multi-Level Security (MLS) systems where sensitive information needs to be shared between security domains and the sensitive exfiltration signatures need to be regularly updated [7], [8]. In particular, our approach enables more secure use of cloud technologies to host sensitive data by reducing the risk of exfiltration.

The contribution of our research is in identifying how to encrypt data for encrypted signature identification, devising a protocol to protect and use sensitive signatures with encryption and provide experimental results on the effectiveness of the approach. Although we focus on bit-string-based signatures comparisons (such as keyword exact-matches in data files' metadata), our approach is generalizable to use signatures to search over malware, text files and audio streams to securely detect these signatures in corresponding data files. These generalized approaches address additional challenges of signature encoding and algorithms design for encrypted signature detection.

An additional novelty of this research is the early application of practical homomorphic encryption technologies. The current state of SHE (and the more general Fully Homomorphic encryption (FHE)) research has primarily focused on either designing new encryption protocols or new tailored SHE and FHE implementations for efficiency improvements. There has been less public discussions integrating homomorphic encryption technologies into broader information ecosystems to provide improved security. Research focusing on how to design algorithms that operate on homomorphic encrypted data has been limited, and similarly few explorations into data structures that would make the integration of homomorphic encryption implementations more effective.

This paper is organized as follows. Section II discusses

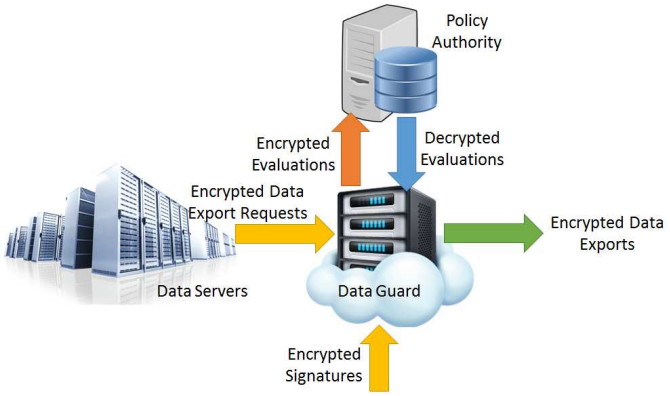

Fig. 1. Encrypted Data Guard Concept of Operations

the overall architecture of our encrypted data guard capability. Section III discusses the security model we address. In Section IV we discuss the salient features of homomorphic encryption. In Section V we discuss the design of the homomorphic encryption signature comparison method. In Section VI we describe the implementation of the filter, concrete parameter selection and experimental evaluation. Section VII discusses related encrypted computing activities and applications. We close the paper in Section VIII which discusses the results, ongoing work and conclusions.

II. Guard Architecture

Our overall, high-level data flows for the encrypted exfiltration monitor and private signature detection can be seen in Figure 1 and an interaction diagram can be seen in Figure 2.

The encrypted data guard architecture consists of a) a protected data center b) the encrypted data guard, c) a policy authority and d) a data recipient.

Prior to use for encrypted data filtering, several offline operations occur:

- The policy authority generates a public key and secret key pair. The public key is widely distributed while the secret key is held by the policy authority. Data can be encrypted with the public key, but can only be decrypted with the secret key.
- The policy authority identifies a set of signatures of interest which should not be found in data files released outside of the protected data center. The policy authority (or approved proxies) encrypt these signatures using the public key and upload the encrypted signatures to the data guard.
- Users of the data center encrypt file data using the public key.

After initialization, the data guard operates as follows as seen in Figure 2:

1) A user requesting to export data submits a request to the data guard with an encryption of data to be exported.
2) The data guard runs our encrypted signature testing mechanism on the encrypted data to test for the presence of the encrypted signature. The data guard does not have access to the secret key and does not perform any decryption.
3) The data guard outputs the result of the encrypted signature testing mechanism for each signature. Each of these outputs is an encryption of a single bit which indicates the presence of a signature in the data file.
4) The encrypted bit is sent to the policy authority.
5) The policy authority decrypts the bit. If the decrypted bit indicates the signature is not in the file, the policy authority sends back a single bit to the guard to approve data export.

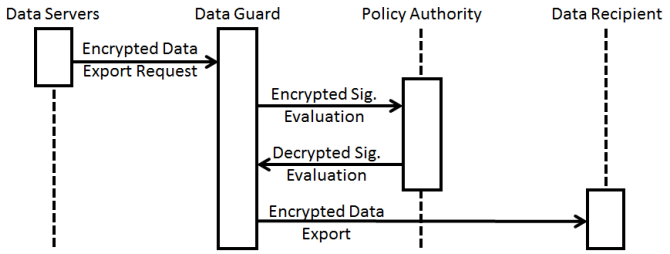

Fig. 2. Encrypted Data Guard Interaction Diagram

In this paper we focus on the novelty of encrypted signature testing with homomorphic encryption to assess either encrypted or unencrypted data flowing across a guard for the presence of encrypted signatures. We do not discuss data integrity, authentication or other supporting security technologies because it is possible to augment the encrypted data guard with these additional security features using standard techniques such as hashing methods to sign the communications between the participants, encrypting the communication from the policy authority to the guard and so on.

III. Security Model

The security goals of the architecture are to prevent the:

- Prohibited export of encrypted data.
- Data guard from having unencrypted access to encrypted data and signatures.
- Policy authority from having access to the encrypted data and signatures.

We allow the data guard and policy authority to have an honest-but curious security model. As such, the data guard follows the directives of the policy authority but has no visibility into the encrypted data exfiltration requests or the signatures. Similarly, the policy authority has no access to the encrypted data exfiltration requests or the signatures, but correctly decrypts the encrypted evaluations and shares the evaluations with the data guard. This honest-but-curious security model protects against eavesdroppers on communication paths, and information loss if any single entity becomes fully compromised.

We admit the possibility that collusion between the data guard and policy authority could allow these participants to gain information about the signatures and the data exfiltration requests. We can limit the practicality of this vulnerability in

practice by limiting communication between the guard and authority.

An important feature of our approach is that the encryption scheme we use is probabilistic. It is possible to use deterministic encryption schemes to evaluate signature comparisons, but deterministic encryption schemes open our approach to a wider array of chosen plaintext attack which would allow eavesdroppers to diagnose past data exfiltration requests by matching new requests to old ones. Because we use a probabilistic encryption scheme, plaintext does not always encrypt to the same ciphertext and we are protected from chosen plaintext attacks and provides much more security than if deterministic encryption methods are used. An important feature of our homomorphic encryption based approach is that even though we focus on exact signature matching, our approach generalizes to support inexact matching such as export approvals based on edit distance from a signature [9].

There is also some likelihood that the data guard or an eavesdropping adversary may be able to infer signatures based on observations of approved and denied exfiltration requests. This is in some sense an unavoidable aspect of the use of data guards - some data will be released and other data will not. Whomever makes data exfiltration requests, or receives the output of data exfiltration requests will be able to observe the success of requests. However, in practical scenarios where data exfiltration requests are made in bulk, and where most requests should be approved due to the lack of pervasive system compromise, adversarial requesters that generate rejected requests are highly likely to be identified and have their data access rights downgraded.

IV. Homomorphic Encryption

By definition, homomorphic encryption enables computing on encrypted data without decrypting. This would allow, for example, the use of encryption of sensitive signatures by host-based systems to perform data guard operations. For example, using a homomorphic encryption scheme, if we have a set of plaintexts $p_1, p_2, \cdots, p_n$, we can securely out-source the evaluation of a function $f(p_1, p_2, \cdots, p_n)$ by:

1) Encrypting $p_1, p_2, \ldots, p_n$ into $c_i = \mathsf{Enc}(p_i, pk)$ for all $i \in \{1, \ldots, n\}$ where Enc is an encryption function and pk is a public key.
2) Converting the function $f(\cdot)$ evaluated over plaintext into a function $F(\cdot)$ evaluated over ciphertext.
3) Evaluating $c' = F(c_1, c_2, \cdots, c_n) = F(\mathsf{Enc}(p_1, pk), \cdots, \mathsf{Enc}(p_n, pk))$.
4) Decrypting $p' = \mathsf{Dec}(c', sk) = \mathsf{Dec}(F(\mathsf{Enc}(p_1, pk), \cdots, \mathsf{Enc}(p_n, pk)), sk)$.

With a homomorphic encryption scheme, if $p = f(p_1, p_2, \cdots, p_n)$, then $p = p'$. Homomorphic encryption is more formally defined in [10], [11].

Figure 3 shows the high-level data flow when using the most general form of HE, called public-key Fully Homomorphic Encryption (FHE) to support arbitrary computing on encrypted data. An FHE client shown at upper left runs on a trusted host and generates public and private key pairs consisting of a public key pk and secret key sk. The public key is shared with a data source (top-right) that encrypts data

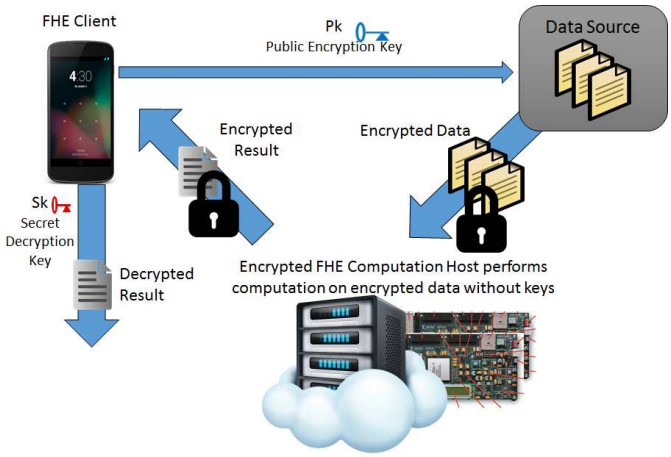

Fig. 3. Fully Homomorphic Encryption

using that public key to generate the encrypted data ciphertexts. The encrypted data is sent to an encrypted FHE computation host, notionally a cloud computing environment, but other host systems, such as embedded devices as hosts are possible. The computation hosts executes a program on the encrypted data without accessing any encryption keys or needing unencrypted data access. The result of this computation is another ciphertext, which we designate as the Encrypted Result in Figure 3. The encrypted result is sent to the FHE client, who decrypts the result using the secret key sk to obtain the plaintext final output.

In our data guard architecture a policy authority hosts the secret key to decrypt ciphertext so the guard does not gain access to decryption keys. Hence, the data guard runs computations on the encrypted data. The output of the guard's encrypted computation is shared with the policy authority who has a decryption key. When the authority uses the decryption key to decrypt the encrypted output, the resulting decrypted data is the same result as if the computation run by the guard had been run on the original data without decryption at the host. Because the result of the string comparison is only available to the guard in an encrypted form, our protocol sends the encrypted result back to the client, where it is decrypted to reveal whether the message should be sent. Thus, our prototype assumes an honest sender and requires an extra round-trip between client and server.

A primary challenge of using homomorphic encryption is the difficulty of translating an arbitrary computation on unencrypted data $f(\cdot)$ into an efficient computation on encrypted data $F(\cdot)$. The primary challenge is that a function $f(\cdot)$ needs to be translated into a function evaluation of a limited set up operations [12], including the EvalMult and EvalAdd operations. If plaintext p_1 and p_2 each encode a bit, and c_1 and c_2 are respectively encryptions of these plaintext, then $c = \mathsf{EvalMult}(c_1, c_2)$ is an encryption of the logical AND of p_1 and p_2, and $c' = \mathsf{EvalAdd}(c_1, c_2)$ is an encryption of the logical XOR of p_1 and p_2. Futher, if we know p_2 is an encoding of the bit 1, then $c^* = \mathsf{EvalAdd}(c_1, c_2)$ is an encryption of the logical NOT of p_1.

A. Homomorphic Encryption Scheme and Security Proof

Although recently discovered HE schemes can support arbitrary computations on encrypted data, many computations are impractically slow [13], but recent very positive results indicate HE designs [14] and implementations [5], [15], [16], [17] are becoming increasingly practical. We show how to use the LTV scheme introduced in [14] to support signature evaluation.

We build on the scheme presented in [5] which is a simplification of the scheme shown in [14]. The basic operations of the scheme are as follows:

- KeyGen: choose a short $f \in R$ such that $f = 1 \bmod p$ and f is invertible modulo q, and a short $g \in R$. Output $pk = h = g \cdot f^{-1} \bmod q$ and $sk = f$.
- $\mathsf{Enc}(pk = h, \mu \in R_p)$: choose a short $r \in R$ and a short $m \in R$ such that $m = \mu \bmod p$. Output $c = p \cdot r \cdot h + m \bmod q$.
- $\mathsf{Dec}(sk = f, c \in R_q)$: compute $\bar{b} = f \cdot c \bmod q$, and lift it to the integer polynomial $b \in R$ with coefficients in $[-q/2, q/2)$. Output $\mu = b \bmod p$.

The security proof of this scheme is derivative of the proof given in [14], so we omit it here for the sake of brevity.

V. Homomorphic Encryption Filter

To design our data guard operation, we translate a plaintext "exact-match" signature comparison operation into a homomorphic evaluation function that performs the same operation on ciphertext. That is, for a signature represented as a set of bits $S = [b_1, b_2, \ldots, b_s]$ and a bit-string of data $D = [b'_1, b'_2, \ldots, b'_s]$ to be compared to S, we say the signature matches the data if

$$\bigwedge_{i=1}^{s} (\neg b_i \oplus b'_i) \tag{1}$$

where $\oplus$ if the logical XOR operation. More informally, we test to see if all pairs of bits between the signature and the data are logically equivalent.

We convert Equation 1 into a homomorphic encryption equivalent as follows. The plaintext $[p_1, p_2, \ldots, p_s, p'_1, p'_2, \ldots, p'_s]$ can each be represented as lists of length-n integers mod p where n and p are configuration parameters. We assign the plaintext $p_i = \{b_i, 0, 0, \ldots, 0\}$, $p'_i = \{b_i, 0, 0, \ldots, 0\}$ and $p^1 = \{1, 0, 0, \ldots, 0\}$ for all $i \in \{1, \ldots, t\}$. We then assign $c_i = \mathsf{Enc}(b_i, pk)$, $c'_i = \mathsf{Enc}(b'_i, pk)$ and $c^1 = \mathsf{Enc}(p^1, pk)$ for all $i \in \{1, \ldots, t\}$. We pre-compute the set of ciphertext $c_i^{\neg} = \mathsf{EvalAdd}(c_i, c^1)$.

For a given a, b where d is chosen such that $d - a$ is the largest power of 2 possible as long as $d < b$, we then define the ciphertext $c^{(a,b)}$ recursively as follows:

$$c^{(a,b)} = \begin{cases} \mathsf{EvalMult}(c^{(a,d)}, c^{(d,b)}) & \text{if } b > a \\ c_a, & \text{if } b = a \\ c^{(b,a)}, & \text{if } b < a \end{cases} \tag{2}$$

Equation 2 recursively defines an algorithm to perform a tree EvalMult operation over the list of ciphertext $[c_a, \ldots, c_b]$. Hence, if $[c_a, \ldots, c_b]$ all represent encryptions of bits, then $c^{(a,b)}$ represent an encryption of the logical AND of all of the bits. Further, the evaluation of $c^{(a,b)}$ induced by the definition is a binary tree evaluation. If we define $c''_i = \mathsf{EvalMult}(c'_i, c_i^{\neg})$, then c''_i is the encrypted evaluation of whether b_i and b'_i are equivalent. We use the evaluation $c''^{(a,b)}$ to define the output of the function $c^* = \mathsf{EvalCompare}(c_1, c_2, \ldots, c_s, c'_1, c'_2, \ldots, c'_s)$ where c^* is an encryption of a single bit which represents whether all pairs (b_i, b'_i) are logically equivalent. Algorithm 1 can be used to compute $c^* = \mathsf{EvalCompare}(c_1, c_2, \ldots, c_s, c'_1, c'_2, \ldots, c'_s)$.

input : Two arrays of ciphertexts
$\{c_1, c_2, \ldots, c_s\}$,$\{c'_1, c'_2, \ldots, c'_s\}$
output: A ciphertext c^*

for $i \leftarrow 1$ **to** s **do**
  $c_i^{\neg} = \mathsf{EvalAdd}(c_i, c^1)$;
  $c''_i = \mathsf{EvalAdd}(c'_i, c_i^{\neg})$;
end
$c^* = c''^{(1,t)}$;

Algorithm 1: EvalCompare() Evaluation Algorithm

As a result of these definitions, $\mathsf{EvalCompare}(c_1, c_2, \ldots, c_s, c'_1, c'_2, \ldots, c'_s)$ represents whether the signature S matches the data D. An important feature of $\mathsf{EvalCompare}(c_1, c_2, \ldots, c_s, c'_1, c'_2, \ldots, c'_s)$ for parameterizing the libraries is the depth of the computation of $\mathsf{EvalCompare}(c_1, c_2, \ldots, c_s, c'_1, c'_2, \ldots, c'_s)$. The depth of computation is how many stages of recursive computations need to be performed in the evaluation of $c^* = c''^{(1,t)}$. Thus $d = \lceil \log_2(t) \rceil$ is the depth of computation needed.

A. Generalizing the Design

Our approach to encrypted exfiltration protection with exact signature matching generalizes beyond exact signature matching. For example, there is a trivial generalization to use the EvalCompare() to search for encrypted strings in corpi of encrypted data. An algorithm for this approach can be seen in Algorithm 2 where a signature string ($\{c_1, c_2, \ldots, c_s\}$) is search for in an encrypted corpus ($\{c'_1, c'_2, \ldots, c'_t\}$) by iteratively performing the EvalCompare() operation over the encrypted corpus.

input : Two arrays of ciphertexts
$\{c_1, c_2, \ldots, c_s\}$,$\{c'_1, c'_2, \ldots, c'_t\}$
output: A ciphertext c^*

$c^{\#} = c^1$;
for $i \leftarrow 1$ **to** $t - s$ **do**
  $c''_i =$
  $\mathsf{EvalCompare}(\{c_1, \ldots, c_s\}, \{c'_i, \ldots, c'_{i+s-1}\})$;
  $c_i^{\neg} = \mathsf{EvalAdd}(c''_i, c^1)$;
  $c^{\#} = \mathsf{EvalMult}(c^*, c_i^{\neg})$;
end
$c^* = \mathsf{EvalAdd}(c^{\#}, c^1)$;

Algorithm 2: EvalCompare() Evaluation Algorithm

A slightly more application-specific generalization arises from the problem of searching for foreign words transliterated

into English. This challenge is especially problematic when searching for transliterated Arabic words where there are multiple mappings from Arabic text to English text based on dialect, accent and even the person or codebook used to perform the transliteration. This problem becomes an issue especially when there are multiple data sources used. This induces a string search optimization that allows for limited wild-cards, such as searching for signatures that allow for the encoding of either "k" or "q" interchangeably, or allow the interchange of encodings of "g" and "j". We can allow this by comparing both of these options in a generalization of Algorithm 1, and then using an inverse of an EvalMult to join the outputs of these results as a kind of homomorphic logical "OR" operation and avoid doing a full signature comparison.

More general than our signature approaches is to use edit distance computations which is also an area of ongoing research in software FHE implementations [9] which we could also use. For example, in the case of mis-spellings, we would be able to filter for approximate matches of words.

VI. Implementation and Experimentation

We implemented our EvalCompare() design in multiple homomorphic encryption libraries and evaluated performance at multiple parameter settings. We implemented three variations of lattice encryption libraries which all support the LTV SHE encryption scheme. These versions were implemented using three different approaches, including:

1) A version implemented in Matlab which runs in the Matlab interpreter.
2) A compiled C/C++ version of the library.
3) A version of the library ported to support acceleration with a Xilinx Virtex 7 FPGA co-processor.

Our designs of these library variants align with the implementations provided in [4], [5], [6].

One of our research goals is to explore the data structure and algorithm needs to support signature matching on encrypted data, hence we do not focus discussion on the low-level implementation details of the homomorphic encryption libraries. However, as a point of comparison, we ran the encrypted string matching algorithm on all three of the homomorphic encryption libraries with a baseline of performance comparison for the parameter settings of $n = 16384$ and $p = 2$, over a range of signature lengths t. The results of the performance comparison can be seen in Figure 4 which shows that our FPGA-accelerated implementation is multiple orders of magnitude faster than the other libraries. As a result of these performance benefits, we focus our design and experimental analyses on the results of using this library.

Following [18], [19], [20], [21], we use the "root Hermite factor" δ as the primary measure of concrete security for a set of parameters. The recent experimental evidence [18] suggests that $\delta = 1.007$ would require roughly 2^{40} core-years on recent Intel Xeon processors to break. Using the estimates from [5], [19], [20], Table I shows how the minimal ring dimension needed varies as a function of the depth of computation needed.

As a result of the analysis of Table I, we decided to simplify our implementation and use a maximum ring dimension of $n = 16384$ for all computations to support $\delta \leq 1.007$ and

TABLE I. Ring dimension n, as a function of depth of computation supported for $p = 2$.

Depth	1	3	5	7	9	11
Min. Ring Dimension n	1024	4096	8192	8192	16384	16384

evaluating matches with signatures of at most 56 bits. With this parameter selection we securely represent ciphertexts as matrices of 64-bit integers, to still execute efficiently in all of our implementations.

We ran further experiments to evaluate the runtime of the EvalCompare operation running on the FPGA implementation to see how the runtime varies for various signature bit lengths for various ring dimensions. The results of these experiments can be seen in Figure 5 which shows this data in a log-log plot. We chose the range of the number of bits in the signature to correspond to the number of bits required to encode ASCII text with 1 to 8 characters. We can see from this plot that runtime grows roughly linearly with the number of bits in the signature across all of the ring dimensions we tested.

As a point of comparison, we also ran experiments to evaluate how runtime is affected by ring dimension. The results of these experiments for the FPGA accelerated hardware can be seen in Figure 6 which shows that how runtime depends on ring dimension when evaluating EvalCompare for various settings of the signature sizes. We see from the graph that performance is super-linear with ring dimension, as would be expected, but less than quadratic asymptotic behavior.

VII. Related Work

Our data guard technique can be thought of as a publish-subscribe system with policy rules to manage subscriptions. This context-aware approach to publish-subscribe systems has an established literature [22], but we focus on a different set of problems where both the subscription policies (i.e., the limitations of exfiltration) and the data flows should be protected [2].

Up to now there have been few feasible approaches to monitor and detect the infiltration or exfiltration with sensitive signatures without requiring visibility into data [2]. There have been prior approaches for Multi-level security management [23]. Some recent practically relevant and applied results in this area include [7], [8], but these prior results do not provide capabilities for encrypted monitoring. In terms of recent research, there have been results that discuss encrypted filtering, [24], [25], [26], [27] and filtering encrypted network traffic [28], [29], but these prior results are not as extensible to more general filtering methods as with the homomorphic encryption capabilities.

VIII. Discussion and Conclusion

We found experimentally that the encoded representation of data can have a very large impact on the runtime of FHE function evaluation in practice. Some early homomorphic systems relied on encoding a single bit of plaintext in each ciphertext. EvalAdd and EvalMult operations were thus simplified into Boolean XOR and AND operations, but offered no computation parallelism. Ciphertext-to-plaintext expansion in such systems is quite large: in one early example of ours,

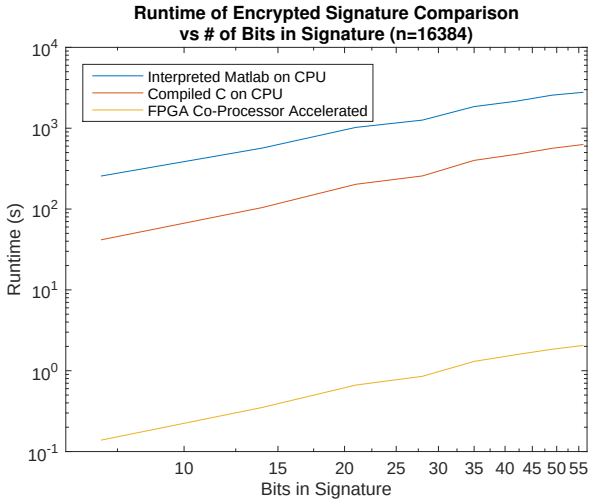

Fig. 4. Encrypted Signature Comparison Runtime vs. Signature Length for Various Computation Devices

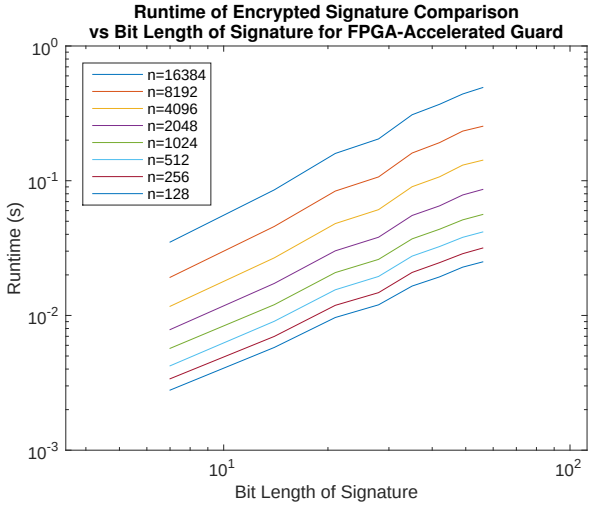

Fig. 5. Encrypted Signature Comparison Runtime vs. Signature Length for Various Ring Dimensions

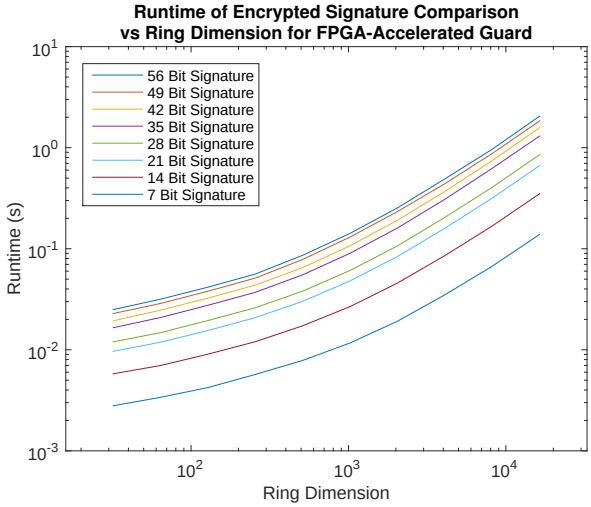

Fig. 6. Encrypted Signature Comparison Runtime vs. Ring Dimension for Various Signature Lengths

the ciphertext expansion ratio was 223. In contrast, modern FHE system encrypts mod p integers ($p > 2$) instead of single bits, and we can leverage Single-Instruction, Multiple Data (SIMD) approaches to pack multiple mod p integers into each ciphertext, thus computing parallel operations on these packed integers. The design of data structures and programs for FHE remains a challenging aspect of applying these technologies. Besides the ciphertext expansion issues, the primitive computation operations, EvalAdd and EvalMult, have uneven performance tradeoffs with EvalAdd operations generally an order of magnitude faster than EvalMult operations.

An important aspect of the scalability of this scheme is that only an encrypted bit is sent to the trusted policy authority. This encrypted bit sharing approach is much more efficient than sending all encrypted data to a policy authority for approval. By computing the encrypted AND of multiple such signature evaluations, we can assess and share the testing for multiple signatures on data files. This reduces bandwidth consumption and prevents the policy authority from having to inspect all data files and only encryptions of single bits need to be communicated.

There are several topics worthy of further exploration in follow-on work. This includes researching how we design a) algorithms and data structures for the challenges of signature comparison testing, b) signature encodings that are efficient on commodity cloud computing hardware like x86-64 compute architectures. c) implementing FHE data structure and algorithms in commodity computing environments and integrate these building blocks into a prototype capability.

IX. Acknowledgment

Partially sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Contract No. FA8750-11-C-0098. The views expressed are those of the authors and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited.)

References

[1] M. I. Center, "Apt1: Exposing one of chinas cyber espionage units," *Mandian. com*, 2013.

[2] D. Parr, "Securing the cloud," *Journal of Information Warfare*, vol. 13, no. 1, pp. 56–69, 2014.

[3] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography*. Springer, 2011, pp. 253–273.

[4] E. Öztürk, Y. Doröz, B. Sunar, and E. Savaş, "Accelerating somewhat homomorphic evaluation using fpgas," Cryptology ePrint Archive, Report 2015/294, Tech. Rep., 2015.

[5] K. Rohloff and D. B. Cousins, "A scalable implementation of fully homomorphic encryption built on NTRU," in *Proceedings of the 2nd Workshop on Applied Homomorphic Cryptography (WAHC)*, 2014.

[6] S. S. Roy, K. Järvinen, F. Vercauteren, V. Dimitrov, and I. Verbauwhede, "Modular hardware architecture for somewhat homomorphic function evaluation."

[7] T. D. Nguyen, M. A. Gondree, D. J. Shifflett, J. Khosalim, T. E. Levin, and C. E. Irvine, "A cloud-oriented cross-domain security architecture," in *MILITARY COMMUNICATIONS CONFERENCE, 2010-MILCOM 2010*. IEEE, 2010, pp. 441–447.

[8] K. Wrona and G. Hallingstad, "Controlled information sharing in nato operations," in *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*. IEEE, 2011, pp. 1285–1290.

[9] J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," IACR Cryptology ePrint Archive, 2015: 132, 2015. To appear in WAHC, Tech. Rep., 2015.

[10] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2009, aAI3382729.

[11] ——, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536440

[12] S. Halevi and V. Shoup, "Algorithms in helib," in *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, pp. 554–571.

[13] 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.

[14] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. ACM, 2012, pp. 1219–1234.

[15] Y. Doröz, Y. Hu, and B. Sunar, "Homomorphic aes evaluation using ntru." *IACR Cryptology ePrint Archive*, vol. 2014, p. 39, 2014.

[16] L. Ducas and D. Micciancio, "Fhew: Bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology–EUROCRYPT 2015*. Springer, 2015, pp. 617–640.

[17] C. Gentry and S. Halevi, "HElib," https://github.com/shaih/HElib, 2014.

[18] Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better lattice security estimates," in *ASIACRYPT*, ser. Lecture Notes in Computer Science, vol. 7073. Springer, 2011, pp. 1–20.

[19] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *CT-RSA*, 2011, pp. 319–339.

[20] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post Quantum Cryptography*. Springer, February 2009, pp. 147–191.

[21] J. van de Pol, "Quantifying the security of lattice-based cryptosystems in practice," in *Mathematical and Statistical Aspects of Cryptography*, 2012.

[22] M. Nabeel, S. Appel, E. Bertino, and A. Buchmann, "Privacy preserving context aware publish subscribe systems," in *Network and System Security*. Springer, 2013, pp. 465–478.

[23] D. E. R. Denning, *Information warfare and security*. Addison-Wesley Reading, MA, 1999, vol. 4.

[24] H. Hacıgümüs, B. Hore, B. Iyer, and S. Mehrotra, "Search on encrypted data," *Secure Data Management in Decentralized Systems, ser. Advances in Information Security*, vol. 33, pp. 383–425, 2007.

[25] R. C. Jammalamadaka, R. Gamboni, S. Mehrotra, K. E. Seamons, and N. Venkatasubramanian, "idataguard: Middleware providing a secure network drive interface to untrusted internet data storage," in *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, ser. EDBT '08. New York, NY, USA: ACM, 2008, pp. 710–714. [Online]. Available: http://doi.acm.org/10.1145/1353343.1353432

[26] G. Miklau and D. Suciu, "Controlling access to published data using cryptography," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003, pp. 898–909. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315528

[27] R. A. 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.

[28] T. Fawcett, "Exfild: A tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic," Ph.D. dissertation, University of Delaware, 2010.

[29] G. Silowash, T. Lewellen, J. Burns, and D. Costa, "Detecting and preventing data exfiltration through encrypted web sessions via traffic inspection," Citeseer, Tech. Rep., 2013.