Privacy-Preserving Character Language Modelling

Patricia Thaine, Gerald Penn
Department of Computer Science, University of Toronto
{pthaine, gpenn}@cs.toronto.edu

CEUR-WS.org/Vol-2335/1st_PAL_paper_10.pdf

Abstract

Some of the most sensitive information we generate is either written or spoken using natural language. Privacy-preserving methods for natural language processing are therefore crucial, especially considering the ever-growing number of data breaches. However, there has been little work in this area up until now. In fact, no privacy-preserving methods have been proposed for many of the most basic NLP tasks.

We propose a method for calculating character bigram and trigram probabilities over sensitive data using homomorphic encryption. Where determining an encrypted character's probability using a plaintext bigram model has a runtime of 1945.29 ms per character, an encrypted bigram model takes us 88549.1 ms, a plaintext trigram model takes 3868.65 ms, and an encrypted trigram model takes 102766 ms.

Introduction

Character-level language models are used in a variety of tasks, such as character prediction for facilitating text messaging. Despite the sensitive nature of the input data, there has been very little work done on privacy-preserving character language models. One example of such work is Apple's use of a version of differential privacy called randomized response to improve their emoji QuickType predictions[1]. Briefly, what users truly type is sent to Apple with a certain probability p and fake input is sent with a probability $1-p$. While this technique can be used for efficiently training a privacy-preserving emoji prediction system, it is easy to see that on its own it would not preserve the privacy of natural language input while preserving its utility.

We propose a privacy-preserving character-level n-gram language model, the inputs to which are entirely accurate and entirely private. We use Homomorphic Encryption (HE) for this purpose. HE has so far been used for a small number of natural language processing and information retrieval tasks. Some of these include spam filtering (Pathak, Sharifi, and Raj 2011), hidden-Markov-model-based spoken keyword spotting (Pathak et al. 2011), speaker recognition (Pathak and Raj 2013), n-gram-based similar document detection (Jiang and Samanthula 2011), language identification (Monet and Clier 2016), keyword search, and bag-of-words frequency counting (Grinman 2016).

[1] https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf

Homomorphic Encryption

Homomorphic encryption schemes allow for computations to be performed on encrypted data without needing to decrypt it.

For this work, we use the Brakerski-Fan-Vercauteren (BFV) Ring-Learning-With-Errors-based fully homomorphic encryption scheme (Brakerski 2012)(Fan and Vercauteren 2012), with the encryption and homomorphic multiplication improvements presented in (Halevi, Polyakov, and Shoup 2018). This scheme is implemented in the PALISADE Lattice Cryptography Library[2]. However, the algorithm we propose can be implemented using any homomorphic encryption scheme that allows for addition and component-wise multiplication in the encrypted domain.

Notation, Scheme Overview, and Chosen Parameters

We will be using the same notation as (Brakerski 2012) and (Fan and Vercauteren 2012), but as we provide only a brief overview of the homomorphic encryption scheme, the specific optimizations introduced in (Fan and Vercauteren 2012),(Halevi, Polyakov, and Shoup 2018) will not be discussed. Let $R = \mathbb{Z}[x]/(f(x))$ be an integer ring, where $f(x) \in \mathbb{Z}[x]$ is a monic irreducible polynomial of degree d. Bold lowercase letters denote elements of R and their coefficients will be denoted by indices (e.g., $\mathbf{a} = \sum_{i=0}^{d-1} a_i \cdot x^i$. $\mathbb{Z}_q$, where $q > 1, q \in \mathbb{Z}$, denotes the set of integers $(-q/2.q/2]$. q is referred to as the ciphertext modulus. An integer n's i-th bit is denoted $n[i]$, from $i = 0$. The secret key is $\texttt{sk} = (1, \mathbf{s})$, where $\mathbf{s} \leftarrow \chi$. The public key is called $\texttt{pk} = ([-(\mathbf{a} \cdot \mathbf{s} + \mathbf{e}]_q, \mathbf{a})$, where $\mathbf{a} \leftarrow R_q$, $\mathbf{e} \leftarrow \chi$.

- $\texttt{Encrypt}(\texttt{pk}, \mathbf{m})$: message $\mathbf{m} \in R_t$, $\mathbf{p_0} = \texttt{pk}[0]$, $\mathbf{p_1} = \texttt{pk}[1]$, $\mathbf{u} \leftarrow R_2$, $\mathbf{e_1}, \mathbf{e_2} \leftarrow \chi$:

$$\texttt{ct} = \Big([\mathbf{p_0} \cdot \mathbf{u} + \mathbf{e_1} + \mathbf{\Delta} \cdot \mathbf{m}]_q, [\mathbf{p_1} \cdot \mathbf{u} + \mathbf{e_2}]_q\Big)$$

- $\texttt{Decrypt}(\texttt{sk}, \texttt{ct})$: $\mathbf{s} = \texttt{sk}$, $\mathbf{c_0} = \texttt{ct}[0]$, $\mathbf{c_1} = \texttt{ct}[1]$.

$$\Big[\Big\lfloor \frac{\mathbf{t} \cdot [\mathbf{c_0} + \mathbf{c_1} \cdot \mathbf{s}]_q}{q} \Big\rceil\Big]_t$$

[2] https://git.njit.edu/palisade/PALISADE

- $\texttt{Add}(\texttt{ct}_1, \texttt{ct}_2)$: $([\texttt{ct}_1[0] + \texttt{ct}_2[0]]_q, [\texttt{ct}_1[1] + \texttt{ct}_2[1]]_q)$
- $\texttt{Add}(\texttt{ct}_1, \texttt{pt}_2)$: $([\texttt{ct}_1[0] + \texttt{pt}_2[0]]_q, [\texttt{ct}_1[1] + \texttt{pt}_2[1]]_q)$
- $\texttt{Mul}(\texttt{ct}_1, \texttt{ct}_2)$: For this paper, we use component-wise multiplication, a simplified description of which is: $([\texttt{ct}_1[0] \cdot \texttt{ct}_2[0]]_q, [\texttt{ct}_1[1] \cdot \texttt{ct}_2[1]]_q)$. The algorithmic details for obtaining this result can be found in (Fan and Vercauteren 2012).
- $\texttt{Mul}(\texttt{ct}_1, \texttt{pt}_2)$: Like with the Add function, it is possible to multiply a ciphertext with plaintext, resulting in: $([\texttt{ct}_1[0] \cdot \texttt{pt}_2[0]]_q, [\texttt{ct}_1[1] \cdot \texttt{pt}_2[1]]_q)$.

Using homomorphic encryption, we can perform linear and (depending on the encryption scheme) polynomial operations on encrypted data (multiplication, addition, or both). We can neither divide a ciphertext, nor exponentiate using an encrypted exponent. We can keep track separately of a numerator and a corresponding denominator. For clarity, we shall refer to the encrypted version of a value $*$ as $E(*)$ and $\times$ to represent Mul.

Optimization: Single Instruction Multiple Data

Single Instruction Multiple Data (SIMD) is explained in (Smart and Vercauteren 2014). Using the Chinese Remainder Theorem, an operation between two SIMD-encoded lists of encrypted numbers can be performed by the same operations between two regularly-encoded encrypted numbers.

Encoding Variables

The very first step in converting an algorithm to an HE-friendly form is to make the data amenable to HE analysis. This includes transforming floating point numbers into approximations, such as by computing an approximate rational representation for them, clearing them by multiplying them by a pre-specified power of 10, and then rounding to the nearest integer (Graepel, Lauter, and Naehrig 2012).

We follow the method suggested in (Aslett, Esperança, and Holmes 2015): choose the number of decimal places to be retained based on a desired level of accuracy ϕ, then multiply the data by 10^{ϕ} and round to the nearest integer. Since we are dealing with probabilities, we do not lose much information when converting, say, 99.6% to 99.

Privacy-Preserving Bigram and Trigram Models

We assume that a user has some sensitive data requiring character-level predictions to be made and that a server has a character-level language model that they do not want to share. We train a bigram model and a trigram model using plaintext from part of the Enron email dataset, which we pre-process to only contain $k = 27$ types (space and 26 lowercase letters). A user's emails are then converted into binary vectors of dimension k.

Bigram Probabilities

For the bigram model, we convert characters into one-hot vectors (e.g., the vector for 'a' has a 1 at index 0). Along with the one-hot vector, k ordered vectors of size k are sent. Each of these vectors are zero-vectors, except for a vector of ones which is at the letter's 'designated index'. Here's a simple example for a language containing $k = 3$ character types. Assume we want to convert letter 'a'; the server is sent two matrices which represent the letter 'a' denoted by $\mathbf{M_{a1}}$ and $\mathbf{M_{a2}}$:

$$\mathbf{M_{a1}} = E([1 \quad 0 \quad 0]), \mathbf{M_{a2}} = E\left(\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\right)$$

Say 'a' is followed by 'b', then the server receives:

$$\mathbf{M_{b1}} = E([0 \quad 1 \quad 0]), \mathbf{M_{b2}} = E\left(\begin{bmatrix}0 & 0 & 0\\ 1 & 1 & 1\\ 0 & 0 & 0\end{bmatrix}\right)$$

The user wants to know how likely is it for 'b' to follow 'a'. The server is able to calculate this like so:

$$\begin{bmatrix}p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33}\end{bmatrix} \times E\left(\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\right)$$

Resulting in:

$$\mathbf{I} = E\left(\begin{bmatrix}p_{11} & p_{12} & p_{13}\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\right)$$

We then take the inner products of each row of $\mathbf{I}$ with $\mathbf{M_{a1}}$ and add them all together. The result, $E(p_{12})$, is sent back to the user, who is able to decrypt it.

Trigram Probabilities

To use the trigram model, we again convert characters into one-hot vectors. Along with them, however, we must send a bit more information. Say we have a trigram probability matrix whose columns are sorted as follows:

$$\begin{bmatrix}c_1c_1\\ c_1c_2\\ c_1c_3\\ c_2c_1\\ c_2c_2\\ c_2c_3\\ c_3c_1\\ c_3c_2\\ c_3c_3\end{bmatrix}$$

If we want to access the row containing the probabilities of c_xc_y, we can find its index with the equation $xt+y$, where t is the number of character types. Let us use 'a' as an example again, with $t = 3$. The user must again send along $\mathbf{M_{a1}} = E([1 \quad 0 \quad 0])$, as well as:

$$\mathbf{M_{a2}} = E\left(\begin{bmatrix}[1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\end{bmatrix}\right), \mathbf{M_{a3}} = E\left(\begin{bmatrix}[1 & 1 & 1]\\ [1 & 1 & 1]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\end{bmatrix}\right)$$

If 'b' follows 'a', the server is then sent $\mathbf{M_{b1}} = E([0 \quad 1 \quad 0])$ and:

$$\mathbf{M_{b2}} = E\left(\begin{bmatrix}[0 & 0 & 0]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\end{bmatrix}\right), \mathbf{M_{b3}} = E\left(\begin{bmatrix}[0 & 0 & 0]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [1 & 1 & 1]\\ [1 & 1 & 1]\\ [1 & 1 & 1]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\\ [0 & 0 & 0]\end{bmatrix}\right)$$

The probability of 'aba', given a trigram probability matrix P_{trigram} can be calculated as follows:

$$\mathbf{I} = \mathbf{P}_{\text{trigram}} \times (\mathbf{M_{a3}} \times \mathbf{M_{b3}})$$

Next, the server take the inner products of each row of $\mathbf{I}$ with $\mathbf{M_{a1}}$ and adds them all together. The result, $E(p_{121})$, is sent back to the user, who is able to decrypt it.

Security

The BGV scheme has semantic security (Albrecht et al. 2018), which means that "whatever an eavesdropper can compute about the cleartext given the ciphertext, he can also compute without the ciphertext" (Shafi and Micali 1984). For our first few experiments, we chose parameters that give over a 128-bit security level the values presented in (Albrecht et al. 2018). This means that it would take over 2^{128} computations to crack the decryption key.

We set the BFV scheme's parameters as follows, to guarantee 128-bit security:

- Plaintext Modulus (t) $= 65537$,
- $\sigma = 3.2$,
- Root Hermite Factor $= 1.0081$,
- $m = 16384$,
- Ciphertext Modulus (q) $= 153249540390512585124987375600208262249902440046968832 1$

To run 256-bit security experiments we set $m = 32768$.

Experiments

The following experiments were run on an Intel Core i-7-8650U CPU @ 1.90GHz and a 16GB RAM.

Plaintext Model, Encrypted Input It takes us 1945.29 ms to output the probability of one encrypted character given the preceding character (also encrypted) and a plaintext character bigram model. It takes us 88549.1 ms to output the probability of one encrypted character given its two preceding characters (both encrypted) and a plaintext character trigram model. These results are based on the parameters listed in Section .

Encrypted Model, Encrypted Input It takes us 3868.65 ms to output the probability of one encrypted character given the preceding character (also encrypted) and an encrypted character bigram model. It takes us 102766 ms to output the probability of one encrypted character given its two preceding characters (both encrypted) and an encrypted character trigram model. These results are also based on the parameters listed in Section .

Additional runtime comparisons are provided in Figure 1 and Figure 2. While the runtime of encrypted models might limit the practicality of their deployment, plaintext models running on encrypted data are practical for deployment, especially when considering predictions parallelizability and the speed ups that better RAM could lead to.

Conclusion and Future Work

We described a method for calculating character-level bigram and trigram probabilities given encrypted data and perform runtime experiments across various security levels to test the scalability of our algorithms. Our next steps will be to adapt this method to word-level language modeling, as well as to create a protocol for training n-gram models on encrypted data.

Figure 1: Runtime comparisons of bigram and trigram predictions for character languages models with 27 to 31 characters across three security levels and between encrypted and plaintext character language models.

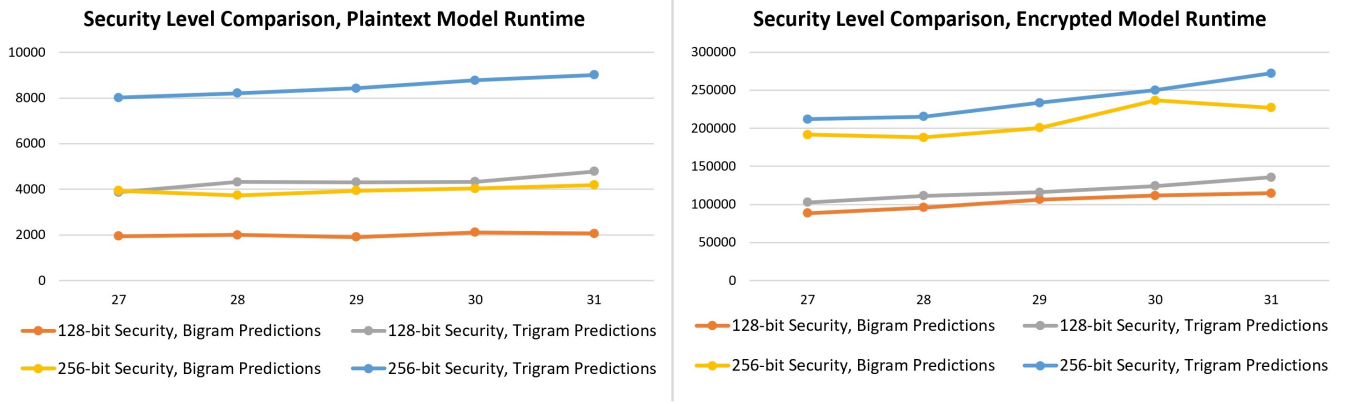

Figure 2: Runtime comparisons of 256-bit security of bigram and trigram predictions for character languages models with 27 to 31 characters.

References

Albrecht, M.; Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Hoffstein, J.; Lauter, K.; Lokam, S.; Micciancio, D.; Moody, D.; Morrison, T.; Sahai, A.; and Vaikuntanathan, V. 2018. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Cambridge MA.

Aslett, L. J.; Esperança, P. M.; and Holmes, C. C. 2015. Encrypted statistical machine learning: new privacy preserving methods. *arXiv preprint arXiv:1508.06845*.

Brakerski, Z. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in cryptology–crypto 2012*. Springer. 868–886.

Fan, J., and Vercauteren, F. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012:144.

Graepel, T.; Lauter, K.; and Naehrig, M. 2012. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, 1–21. Springer.

Grinman, A. J. 2016. *Natural language processing on encrypted patient data*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Halevi, S.; Polyakov, Y.; and Shoup, V. 2018. An improved rns variant of the bfv homomorphic encryption scheme. *IACR Cryptology ePrint Archive* 2018:117.

Jiang, W., and Samanthula, B. K. 2011. N-gram based secure similar document detection. In *IFIP Annual Conference on Data and Applications Security and Privacy*, 239–246. Springer.

Monet, N., and Clier, J. 2016. Privacy-preserving text language identification using homomorphic encryption. US Patent 9,288,039.

Pathak, M. A., and Raj, B. 2013. Privacy-preserving speaker verification and identification using gaussian mixture models. *IEEE Transactions on Audio, Speech, and Language Processing* 21(2):397–406.

Pathak, M. A.; Rane, S.; Sun, W.; and Raj, B. 2011. Privacy preserving probabilistic inference with hidden markov models. In *ICASSP*, 5868–5871.

Pathak, M. A.; Sharifi, M.; and Raj, B. 2011. Privacy preserving spam filtering. *arXiv preprint arXiv:1102.4021*.

Shafi, G., and Micali, S. 1984. Probabilistic encryption. *Journal of computer and system sciences* 28(2):270–299.

Smart, N. P., and Vercauteren, F. 2014. Fully homomorphic simd operations. *Designs, codes and cryptography* 71(1):57–81.