Homomorphic Cryptosystems

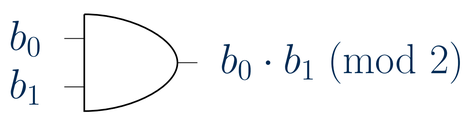
Jeremy Caci, Ali Hajy, Jonah Jolley
Clark Rinker, Philip Robinson
Advisor : Dr. David Bover
Computer Science Department

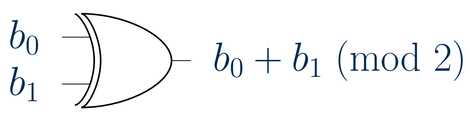
Abstract

We present a formal inquiry of the first fully homomorphic encryption scheme proposed by Craig Gentry in 2008. Moreover we introduce the first implementation of this cipher in the Python programming language. A fully homomorphic encryption scheme enables the execution of arbitrary operations on encrypted data without the decryption key which, put simply, allows for a third party to store and manipulate sensitive information without the ability to interpret it.

Our investigation includes a high level synopsis of the mathematics involved in fully homomorphic encryption, a complete demonstration of our implementation in Python, and finally a overview of the space and asymptotic time complexities of the proposed system.

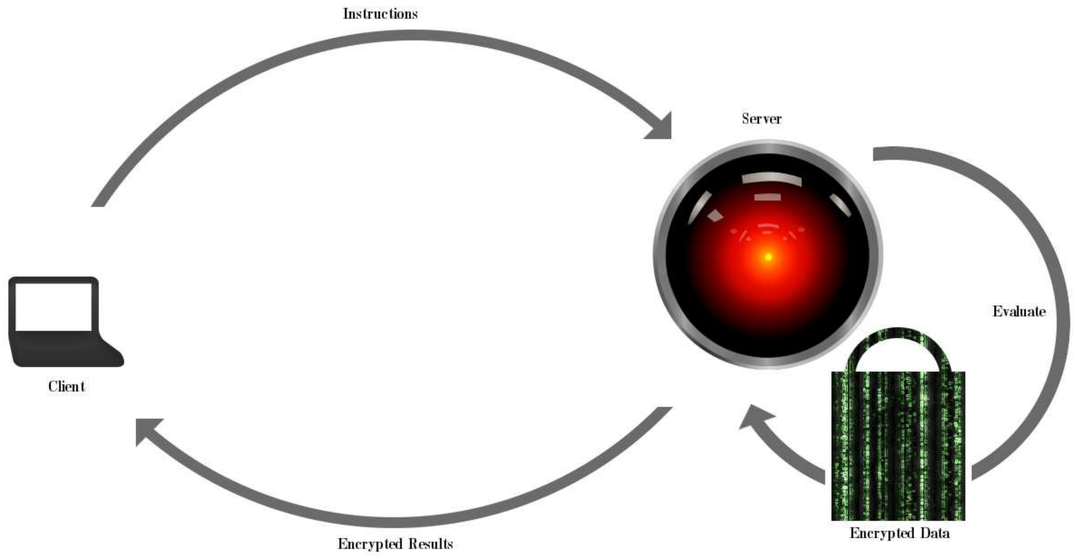
Components





$$c_i \cdot c_j = (p \cdot q_i + 2 \cdot n_i + b_i) \cdot (p \cdot q_j + 2 \cdot n_j + b_j) = p \cdot \hat{q} + 2 \cdot \hat{n} + (b_i \cdot b_j)$$

$$c_i + c_j = (p \cdot q_i + 2 \cdot n_i + b_i) + (p \cdot q_j + 2 \cdot n_j + b_j) = p \cdot \hat{q} + 2 \cdot \hat{n} + (b_i + b_j)$$



KeyGen

Uses λ to generate two keys, a public key that is available to everyone, and a secret key that is only available to the client to be used in decryption.

Encrypt

Maps a plaintext message to a ciphertext using the public key.

Decrypt

Maps a ciphertext back to its original message using the secret key.

Recrypt

Encrypts a previously encrypted ciphertext using the public key, which effectively reduces the space and increases the integrity of the data.

Evaluate

Given the public key and a set of arbitrary operations, this function will execute each instruction over the entire ciphertext and return the encrypted results.

Security

Given a security parameter λ there exists a private key space of

$$2^{(\lambda^2-2)} - (2^{\lambda^{2-1}}) - 1$$

The cryptographic hint uses the subset sum problem over a sparse subset. A brute force attack on this scheme requires the space. With $\lambda = 8$, $\beta = 8^5$, and $\alpha = \frac{8^5}{2}$, an attacker would need to calculate $\binom{\beta}{\alpha}$ subset sums. This would take $2 \cdot 10^{133715000000000000000}$ floating point operations; at a petaflop of computational power this would take $2 \cdot 10^{133715}$ years to brute force the private key.

Limitations

Space Complexity

This cryptographic model has not yet been refined to a practical level. Encrypting data bitwise under these methods, such that the cipher text is cryptographically secure requires a great degree of noise/bloat to hide the parity of our unencrypted message bidgit.

	bit-size	encrypted volume	$\lambda = 8$	$\lambda = 16$	$\lambda = 64$
bidgit	1	$\approx \lambda^7$	$\approx$ 256 KB	32 MB	512 GB
"cadadr"	6 Bytes		$\approx$ 12 MB	1.5 GB	24 TB
Declaration of Independence	8.5 KB		$\approx$ 17 GB	2.1 TB	34 PB

Next Steps

The purposed system is effective but not efficient. In order to achieve enough speed up and make the system realistic we propose to parallelize server side operations across the Hadoop distributed computing system.

Currently our proposed system uses elementary school style long multiplication, an operation that is N^2 for the number of digits in the operands. Even with Karatusba's algorithm $\theta(n \lg n)$ implemented in python the system does not achieve a sufficient speed up. We plan to implement the multiplications on the video cards of our lab using Pycuda and Fast DFT. This parallelization would achieve a speed not normally usable in python.

Finally in order to shrink space complexity we plan to explore using n-ary systems rather than a base-2 implementation. This research would require a more formal address of our co-sets and ideals.

Conclusion

Our Implementation

We were successful in being the first group to implement a fully homomorphic encryption system using the Python programming language. The concerns about the efficiency of Python were surmounted using alternative optimized computation libraries, namely PyPy. We addressed the time and space limitations of this encryption system by utilizing a distributed file system tool called Hadoop and a cluster of powerful computers available to us in Western's Computer Science Department.

Feasibility

Even with the resources available to us, the overhead of this encryption scheme make a contemporary model impractical. The space required to encrypt and store a data set with a sufficient security parameter is unreasonable. However, the concept of a fully homomorphic encryption scheme is still in its infancy, and since Craig Gentry's initial dissertation multiple papers have been published containing new and exciting optimizations to the original algorithm. This combined with the rate at which modern hardware is advancing might make the idea of a functional fully homomorphic system possible in the future.

Bibliography

[1] Craig Gentry. Computing arbitrary functions of encrypted data. Communications of the ACM 53.3 (2010): 97-105. Web, 2010.

[2] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. Manuscript, 2010.

[3] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009.