Computing Arbitrary Functions of Encrypted Data

Klev

September 14, 2015

- Fully homomorphic encryption scheme.
- Keep data private, only people with access can decrypt it.
- Still, possible to compute on the encrypted data.
- Cloud computing compatible with privacy

- Fully homomorphic encryption
- Fully: No restrictions on the operations that can be performed
- Send description of function f to the server. Server computes the function on the encrypted data. Returns encrypted data. User decrypts.
- Useful whenever the response can be encrypted.
- Alice and her jewelry store.

1 Homomorphic Encryption: Funcitonality

Three algorithms: $KeyGen$, $Encrypt$, $Decrypt$

Symmetric encryption scheme: Key used for both encryption and decryption

- Asymmetric encryption scheme: Public key for encryption, Private key for decryption
- HE can be either symmetric or asymmetric. Focus on asymmetric.
- Fourth algorithm: *Evaluate*. A set of permitted functions F_e. *Evaluate* can handle functions in F_e. Others are undefined.

2 Requirements

- Require that decrypting c (output from Evaluate) takes the same amount of time as decrypting c_1 (output from Encrypt).
- c is the same size as c_1
- "Compact ciphertext requirements"

- Size of c and the time required to decrypt it is independent of f

- Fully homomorphic if it can handle all functions, has compact ciphertexts and *Evaluate* is efficient.

- The work required by Alice to extract jewelry has no connection to the time spent by workers to assemble the piece.

- Measure complexity of function f: Running time T_f and size S_f of a boolean circuit.

- Break down the computation of f into AND, OR and NOT gates.

- Obtain fully homomorphic encryption by operating on ciphertexts using add, subtract and multiply.

- If encrypted, random-access can not speed up the work. *Evaluate* must touch all points. Runtime is at least linear.

- Not touching them would leak information, saying that they are irrelevant for function f.

- Trade off: the data is contained in 1% of my area. Reduce computation by factor 100.

- Should be able to specify the amount of data received from query. Unavoidable with padding and truncating.

3 Homomorphic Encryption: Security

- Semantic security against chosen-plaintext attacks (CPA)

- Given c and two messages m_1 and m_2, it is *hard* for an adversary to determine which plaintext the ciphertext decrypts to.

- "Hard": Guesses correctly with probability $1/2 + e =$ Advantage

- Deterministic: Same plaintext encrypts to same ciphertext every time.

- The scheme must be probabilistic; a plaintext encrypts to multiple ciphertexts.

- Malleability weakens deterministic schemes, not semantically secure.

4 A Somewhat Homomorphic Encryption Scheme

- First as a symmetric encryption scheme.

- $N = \lambda$, $P = \lambda^2$, $Q = \lambda^5$

- KeyGen: Key is random P-bit odd integer p

- Encrypt(e,m): $m' =$ random N-bit number such that $m' = m \bmod 2$. $c \leftarrow m' + pq$

- Decrypt(p,c): $(cmodp)mod2$, where $cmodp$ is the integer c' in $(-p/2, p/2)$ such that p divides $c - c'$.

- $cmodp$ is the noise associated to a ciphertext. Distance to nearest multiple of p.

- Consider $Mult_e$: $c = c_1 * c_2$. Noise to $c_i = m'_i$. We have that $c = m'_1 * m'_2 + pq'$

- Multiplication tends to increase the noise faster than addition and subtraction.

- c_1 and c_2 with k_1 and k_2-bit noises = roughly $(k_1 + k_2)$-bit noises.

- The functions that can be handled are those where $|f^{\dagger}(a_1, ..., a_t)|$ is always less than $p/2$.

5 Bootstrappable Encryption

- Work on box 1, put it inside box 2 which contains key 1, open box 1 and continue work through box 2.

- Somewhat homomorphic encryption: Supports Add, Subtract, Multiply, and a limited set of operations, until the noise gets too large.

- Key for box i represents an *encrypted secret decryption key.*

- Most important function: The decryption function (Open up the inner box to continue work)

- Decrypts itself $\rightarrow$ bootstrappable

5.1 Boostrappable to Fully Homomorphic

- Suppose e can handle decryption, ADD, SUB and MULT.

- If these four algorithms are in f_e, we can construct an encryption scheme that is fully homomorphic.

- $Recrypt_e(pk_2, D_e, sk_1, c_1)$ Recrypt outputs an encryption of m, but under the new key pk_2. The message is encrypted twice under different keys pk_1 and pk_2.

- Can "peel off" the layers like an onion, but the *evaluate* algorithm "opens" the inner encryption. (Like in the box example).

- Decryption the inner layer removes noise. Using *Evaluate* with pk_2 adds new noise. If the added noise is less than what we remove in the decryption, we've made progress.

- Public key consists of a sequence of public keys $(pk_1, pk_2, ..., pk_n$ encrypted under secret keys sk_{i+1}

5.2 Circular Security

- Safe to reveal the encryption of a secret key s_k under its own associated public key p_k.

5.3 Greasing the Decryption Circuit

- $m \leftarrow (cmodp)mod2 == m \leftarrow LSB(c) \oplus LSB(\lfloor c/p \rceil)$

- If not $\lfloor c/p \rceil$ is complicated, then e is bootstrappable and FHE is possible. Not possible with the current scheme. Let's create $e*$

- Replace e's decryption function, which multiples two long numbers, with a decryption function that adds a fairly small set of numbers.

5.3.1 Transformation

- KeyGen: Run the algorithm to obtain p_k and $s_k(s_k$ is an odd integer p.

- Generate a subset from s_k of rational numbers that sums up to roughly $1/p$. s_k will then be the subset y of the numbers encoded as a vector. $p_k* \leftarrow (pk, y)$

- This is a "hint" about the secret key. Adds the "grease" to the system. Revealing this obviously impacts security.

- Encrypt: Run Encrypt(p_k,m) to obtain ciphertext c. For $i \in \{1, ..., \beta\}$, set $z_i \leftarrow c * y_i mod 2$. The ciphertext $c*$ is now c and $z = \{z1, ..., z_\beta\}$

- The hint is used to postprocess the ciphertext. The idea is to leave less work for the Decrypt algorithm.

- Used in 'Server-aided cryptography'.

- The hint is statistically dependent on s_k, but enough for the server to decrypt efficiently on its own.

- In our case: The encrypter or evaluator plays the role as server

- Decrypt: Output $LSB(c) \oplus LSB(\lfloor \sum s_i z_i \rceil)$

- Works because $\sum s_i z_i = \sum c * s_i y_i = c/p mod 2$

- Important: Replace the multiplication of c and $1/p$ with a summation that contains only α nonzero terms.

- Add: (p_k*, c_1*, c_2*). Extract c_1 and c_2 from c_1* and c_2*. Run $c \leftarrow Add(p_k, c_1, c_2)$. The result consist of ciphertext $c*$. $c* = c$ and the result of postprocessing c with y.

5.4 How to Add Numbers

- Let us consider the computation of $\sum s_i z_i$.

-

5.5 Security of the Transformed Scheme

- The encryption key contains a hint about the secret p.