Tools for Computing on Encrypted Data

Scribe: Pratyush Mishra

September 29, 2015

1 Introduction

Usually when analyzing computation of encrypted data, we would like to have three properties:

1. **Security:** Ideally, we would like to attain semantic security.
2. **Functionality:** We would like to be able to compute any function on encrypted data that we can on the corresponding plaintext.
3. **Performance:** The performance of our scheme should be comparable to that of computation on unencrypted data.

However, achieving all three of these properties optimally is a difficult proposition. Indeed, just achieving the first two simultaneously was an open question for 30 years, and was only answered in the affirmative in 2009 by Craig Gentry, with his scheme for **Fully Homomorphic Encryption** (FHE).

2 Fully Homomorphic Encryption

The idea of fully homomorphic encryption was first proposed by Rivest, Adleman and Dertouzos in 1978 [RAD78]. They formally describe what a fully homomorphic encryption scheme should look like and what properties it should have. However, their idea was only realized in 2009 by the seminal work of Craig Gentry [Gen09]. Here we describe a simplified version of FHE over the integers.

2.1 Key Algorithms

A general FHE scheme is a public key encryption scheme consisting of four main algorithms:

- **Key Generation:**

  This function generates a public-private keypair.

  $\mathsf{KeyGen}(1^k) \to \mathsf{pk}, \mathsf{sk}$

- **Encryption:**

  $\mathsf{Enc}(1^k, \mathsf{pk}, m) \to c$

- **Decryption:**
  $\mathsf{Dec}(1^k, \mathsf{sk}, c) \to m$
- **Evaluation:**
  This functions takes in n ciphertexts, and a function that is to be evaluated on the underlying plaintexts, and returns the encryption of said evaluation.
  $\mathsf{Eval}(\mathsf{pk}, c_1, \dots, c_n, f) \to \mathsf{Enc}(1^k, \mathsf{pk}, f(m_1, \dots, m_n))$

2.2 Construction for Symmetric Key Encryption

For the sake of exposition, we will use a private key encryption scheme constructed below, instead of the public key encryption scheme we described above. Let the parameters of the system be: $N = k, |p| = k^2, |q| = k^5$. The corresponding versions of the above algorithms are:

- $\mathsf{KeyGen}(1^k) \to \mathsf{sk} = p$, where p is an odd integer.
- $\mathsf{Enc}(1^k, \mathsf{sk}, m) \to m' + pq$
  where $m' \overset{\$}{\leftarrow} \{0,1\}^N$ s.t. $m' = m \pmod 2$, and $q \overset{\$}{\leftarrow} \{0,1\}^{k^5}$
  Here adding pq adds noise to m', thus obscuring the parity of the message.
- $\mathsf{Dec}(1^k, \mathsf{sk}, c) \to (c \bmod p) \pmod 2$
  This works because c is a near multiple of p, since $|m'| \ll |q|$. Hence we have very low possibility of overlap. Modding out by p removes the noise we added during the encryption process, and thus allows us to recover the parity of the original message.

2.2.1 Homomorphism

The term homomorphism has a formal mathematical definition. We can informally view Enc above as a homomorphism from the ring of plaintexts to that of ciphertexts, and Dec as the inverse homomorphism. However, this is not actually a homomorphism, because homomorphisms must map the identity element in one ring to that in the other, which would break the security of our scheme.

So for us the homomorphic property of the scheme simply means that an operation on ciphertexts corresponds to some operation on the underlying plaintexts. We claim that the above encryption scheme is homomorphic:

1. **Addition mod 2/XOR**:
   To add the underlying plaintexts, we simply add the corresponding ciphertexts:
   $$c_1 + c_2 = m_1' + pq_1 + m_2 + pq_2 = (m_1' + m_2') + p(q_1 + q_2)$$
   Clearly, $|c_1| + |c_2| \ll |p|$, so the decryption will still work as expected.

2. **Multiplication mod 2**:

   Similar to above, to multiply plaintexts we multiply the correspondng ciphertexts:

$$\begin{aligned} c_2 \cdot c_2 &= (m_1' + pq_1)(m_2' + pq_2) \\ &= m_1' \cdot m_2' + p(q_2 m_1 + m_2' q_1 + pq_1 q_2) \end{aligned}$$

   Here $|m_1' \cdot m_2'| < p$. One caveat that we notice is that we cannot keep performing multiplications endlessly, because eventually $|m_1' \cdot m_2'| > p$ for some such iteration.

2.2.2 Security

The claim is that finding the secret key is roughly as difficult as solving the **Approximate GCD problem**:

1. Sample s_i, p, q_i of size k, k^2, k^5 bits respectively.
2. Then no PPT adversary can
   - Compute p given $x_i = s_i + pq_i$.
   - Distinguish $(s_i \mod 2, s_i + pq_i)$ from $(b_i, r_i + pq_i)$.

Thus we have semantic security. To also obtain hiding of the circuit, we let the function given to Eval be a universal circuit, and pass the actual function to be evaluated as another ciphertext.

2.2.3 Fully Homomorphic Encryption

The above encryption scheme only supports a limited number of additions and multiplications before the noise overwhelms the message, rendering the ciphertext undecryptable. To overcome this, we need a way to periodically "refresh" the ciphertext and reduce the noise so that we can continue operations. We can achieve this via the Recrypt operation below:

$\mathsf{Recrypt}(1^k)$:

1. $c_2 = \mathsf{Enc}(1^k, \mathsf{sk}_2, c_1)$
2. $\mathsf{Eval}(\mathsf{sk}_2, c_2, \mathsf{Dec}_{p_1})$

This privately decrypts the ciphertext c_1, and the result is already encrypted under the secret key sk_2, so the "server" never learns the value of the ciphertext.

3 Garbled Circuits

Garbled Circuits are another cryptographic primitive introduced in the context of secure two party computation by Andrew Yao [Yao86]. Formally, we have:

Definition 1 *Given a circuit C that takes inputs $\vec{m} = m_1, \ldots, m_n$, we can define a garbled circuit as the output of a function* Garble, *defined as:*

$$\mathsf{Garble}(C) = GC,\ \mathsf{sk} = ((L_1^0, \ldots, L_n^0), (L_1^1, \ldots, L_n^1))$$

where $\mathsf{Enc}(\vec{m}, \mathsf{sk}) = (L_0^{m_0}, \ldots, L_1^{m_1})$*. To evaluate this garbled circuit, we have a function* $\mathsf{Eval}(GC, \vec{c}) = C(\vec{m})$*, that is, it is a function whose output on a ciphertext input is the same as the output of the original circuit* C *on the corresponding plaintext input. Optionally, one can require that the result be encrypted so that the adversary cannot read it.*

Garbled Circuits offer almost ideal security guarantees:

- **Privacy:** The adversary learns nothing other than $C(m)$.
- **One-time:** Reusing the same garbled circuit breaks all privacy guarantees.
- **Authenticity:** Attacker cannot convince client of incorrect result.

In addition to the above strong security guarantees, garbled circuits are also practical today, with creation and evaluation both being done on the order of a few milliseconds today. However, this practicality depends on the size of the circuit corresponding to the function being computed.

4 Functional Encryption

A general functional encryption scheme is a public key scheme where the holder of the master secret key sk can generate and distribute a "function secret key" (sk_f) that allow anyone that has this key and the ciphertext $\mathsf{Enc}(\mathsf{pk}, x) = c$ to decrypt the ciphertext with this key and obtain $\mathsf{Dec}(\mathsf{sk}_f, c) = f(x)$. General functional encryption schemes have only been realized recently in [GKP+13, GGH+13].

Definition 2 *A functional encryption scheme is a 4-tuple of algorithms:*

- $\mathsf{Setup}(1^k) \to \mathsf{sk}, \mathsf{pk}$
- $\mathsf{Enc}(1^k, \mathsf{pk}, x) \to c$
- $\mathsf{KeyGen}(1^k, \mathsf{sk}, f) \to \mathsf{sk}_f$
- $\mathsf{Dec}(1^k, \mathsf{sk}_f, c) \to f(x)$*, where* $c = \mathsf{Enc}(1^k, \mathsf{pk}, x)$

This primitive offers strong privacy guarantees in that the adversary learns nothing about the message m other than $f(m)$. However, it is highly impractical at the moment, with no known implementations of general functional encryption.

While at the surface the "leaking" nature of functional encryption might make it seem useless, it actually engenders many applications. For instance, a spam filter can check whether incoming encrypted mail is spam or not, while simultaneously maintaining the security offered by the encryption scheme.

References

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proc. STOC 2009*, volume 9, pages 169–178. ACM, 2009.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *Proc. FOCS 2013*, pages 40–49. IEEE, 2013.

[GKP$^+$13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proc. STOC 2013*, pages 555–564. ACM, 2013.

[RAD78] Ronald L Rivest, Leonard Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *Proc. FOCS 1986*, pages 162–167. IEEE, 1986.