PRIVACY HOMOMORPHISMS: APPROACHES, IMPLEMENTATION AND APPLICATIONS

The concept of privacy homomorphism was first introduced by Rivest, Adleman and Dertouzos in 1978. A privacy homomorphism, or homomorphic encryption, is an encryption transformation which allows the encrypted data to be operated on without knowledge of the decryption function. A privacy homomorphism allows the data handler to perform computations on the encrypted data and the data owner gets the computing result through decryption.

The homomorphic properties exist in various cryptosystems. We classify all privacy homomorphism systems into three types: homomorphic public-key cryptosystems, algebraic privacy homomorphisms and fully homomorphic encryptions.

Homomorphic public-key cryptosystems: there are many public-key cryptosystems satisfying homomorphic properties. Popular public-key systems, such as RSA, ElGamal, Paillier, etc. are all homomorphic. For example, RSA is multiplicatively homomorphic; ElGamal is also multiplicative; and Paillier is additively homomorphic. There are many other homomorphic public-key systems which are not listed here. But none of the public-key systems known so far support both homomorphic addition and multiplication. So homomorphic public-key systems can only be used in very simple homomorphic calculations.

Algebraic privacy homomorphisms: privacy homomorphisms which support both homomorphic additions and multiplications are usually called algebraic privacy homomorphisms. So far all known algebraic privacy homomorphisms are symmetric-key cryptosystems. Such systems were first proposed by Rivest, Adleman and Dertouzos in their paper in which the concept of privacy homomorphism was first proposed. However, their systems were proved to be insecure. Doming-Ferrer proposed two such privacy homomorphisms. But they were gradually broken by known-cleartext attacks. In fact, algebraic privacy homomorphisms are generally weak in security. So they are applicable in situations where only low security level is required.

Fully homomorphic encryptions: Fully homomorphic encryption is a very new research area. In 2009, Craig Gentry proposed the first fully homomorphic encryption scheme based on ideal lattices, which is a break through. After Gentry, many improvements on his scheme were made by different researchers and there were several new fully homomorphic encryptions were proposed. Brakerski and Vaikuntanathan showed another well-known fully homomorphic encryption based on LWE. The emergence of fully homomorphic encryptions is a milestone in the research history of privacy homomorphisms. However, the current fully homomorphic encryptions are not usable in the real world, due to the lack of efficiency.

Among the above three types, we think the algebraic privacy homomorphism is the most practical solution for homomorphic computing. The public-key systems offer too limited operations and the fully homomorphic encryptions are not practical at all. So we focus on symmetric-key algebraic privacy homomorphisms in this paper. And the term *privacy homomorphism* used in this paper all refer to this type of homomorphic systems.

There are some facts about the security of privacy homomorphisms. It is proved that, if a privacy homomorphism preserves order when encrypting, it is insecure against a ciphertext-only attack. And if a privacy homomorphism is additive, it must be insecure again a chosen-cleartext attack. So an additive privacy homomorphism can at most be secure again a known-cleartext attack.

One famous example of privacy homomorphisms is the one based on the Chinese Remainder Theorem: select two secret large primes p, q, and let $n = pq$. Given cleartext $x \in \mathbf{Z}_n$, the encryption is to compute $(x \bmod p, x \bmod q)$ as the ciphertext. And given the ciphertext vector, if p, q are known, use the Chinese remainder theorem to compute cleartext x, which acts as the decryption. This scheme satisfies both additive and multiplicative homomorphisms. And the data handler computes the sum, difference, or product of two ciphertexts by performing the operations componentwise , modulo n. Without knowing p, q, one is not able to decrypt the ciphertexts. If the attacker wants to get p, q from n, he need to factor n;however, this is intractable when n is large. This system is probably the first well-known privacy homomorphism. But it is shown that it can be broken by a known-cleartext attack.

Doming-Ferrer enhanced the above system by making it more complex. The rough idea is: first break the cleartext x into several parts $x_1, x_2, ..., x_d$, which sum up to x itself. Multiply each component x_i by polynomial r_p^i and r_q^i where r_p, r_q are random parameters. Then modulo the two polynomials with p and q respectively. So the ciphertext appears in the form $([x_1 r_p \bmod p, x_1 r_q \bmod q], ..., [x_d r_p^d \bmod p, x_d r_q^d \bmod q])$. The decryption just takes inverted steps: for the ith component of the ciphertext vector, multiply it by r_p^{-i} and r_q^{-i}, and get $[x_i \bmod p, x_i \bmod q]$; then apply the Chinese Remainder Theorem and recover x_i; sum up all $x_1, ..., x_d$ and recover cleartext x. Only $n = pq$ is made public, p, q, r_p, r_q are all private. The system can deal with homomorphic additions, subtractions and multiplications. Notice that the ciphertext is actually two d-degree polynomials: $x_1 r_p + x_2 r_p^2 ... + x_d r_p^d \bmod p$ and $x_1 r_q + x_2 r_q^2 + ... + x_d r_q^d \bmod q$. So the operations on the ciphertext are just like dealing with polynomial additions, subtractions and multiplications. This system was claimed to be able to withstand known-cleartext attacks, however, later researches show that it still can be broken.

The above two privacy homomorphisms both support homomorphic additions, subtractions and multiplications, i.e. homomorphic operations over rings. However, they do not support homomorphic operations over fields. In fact, seldom privacy homomorphisms support all operations over fields. In this thesis, we propose a new privacy homomorphism, which support all field operations. The approach is rather simple: Choose two secret large primes p, p', where $p' > p$, and let $n = pp'$. For a given cleartext $x \in \mathbf{Z}_p$, compute $y = x^p \bmod n$ as the ciphertext, which is the encryption. And for a given ciphertext $y \in \mathbf{Z}_n$, recover the cleartext by computing $x = y \bmod p$, which is the decryption. If p is unknown, it is not able to recover the cleartext. Since only n is public, if the attacker tries to get p from n, he needs to do factorization. All field operations, i.e. addition, subtraction, multiplication and multiplicative inverse, are available, which is easy to prove by Fermat's Little Theorem. As to the security, unfortunately, the system can only withstand known-cleartext attacks and can be broken by known-cleartext attacks. But we think there are possible ways to make it secure against known-cleartext attacks. One possible way is to combine this scheme with other homomorphic transformations. For example, one may first perform the Chinese Remainder Theorem to the cleartext, and then apply this scheme; or one may first ho-

momorphically map the cleartext to another form (e.g. map integers to polynomials), and then use this scheme to encrypt. In this way, the security will be enhanced, but it will lower the efficiency.

Through the study of privacy homomorphisms, it seems to be a fact that, the more homomorphic operations a privacy homomorphism supports, the lower its security level will be; and if the security is strengthened, the efficiency will be sacrificed. To some degree, the homomorphic property seems to be incompatible with a crypto algorithm. Because to encrypt something means to make it diffused and confused, the aim is to remove order; while a homomorphism means to preserve some sort of order in the ciphertext. It is a dilemma! Therefore, a good privacy homomorphism is difficult to find; and the biggest challenge in the research of privacy homomorphisms is to deal with this dilemma.

Typical applications of privacy homomorphisms include computing delegation and data delegation. Computing delegation usually happens when a (small) company or organization wants to use external facilities to do calculations on the confidential data. In this case, the data owner sends out the encrypted version of the confidential data; the data handler uses homomorphic operations to do calculations on the encrypted data and returns the encrypted result; the data owner receives the encrypted result and decrypts it to get the real result. In this process, the data handler gets into contact only with encrypted texts, and therefore he can launch only ciphertext-only attacks. So privacy homomorphisms used in this case are only required to withstand ciphertext-only attacks. In the case of data delegation, the data handler not only performs operations on the encrypted data but also requires the decrypted result of the computation, so he possesses cleartext-ciphertext pairs. Therefore, privacy homomorphisms used in this case are required to withstand known-cleartext attacks.

In future work, the creation of a good algebraic privacy homomorphism, i.e. a homomorphic cryptosystem which is both secure and practical, is still a very prominent problem in the research of homomorphic cryptography.