DOI:10.1145/1666420.1666445

Technical Perspective
A First Glimpse of Cryptography's Holy Grail

By Daniele Micciancio

WE ALL KNOW how to protect our private or most valuable data from unauthorized access: encrypt it. When a piece of data M is encrypted under a key K to yield a ciphertext $C=Enc_K(M)$, only the intended recipient (who knows the corresponding secret decryption key S) will be able to invert the encryption function and recover the original plaintext using the decryption algorithm $Dec_S(C)=Dec_S(Enc_K(M))=M$.

Encryption today—in both symmetric (where $S=K$) and public key versions (where S remains secret even when K is made publicly available)—is widely used to achieve confidentiality in many important and well-known applications: online banking, electronic shopping, and virtual private networks are just a few of the most common applications using encryption, typically as part of a larger protocol, like the TLS protocol used to secure communication over the Internet.

Still, the use of encryption to protect valuable or sensitive data can be very limiting and inflexible. Once the data M is encrypted, the corresponding ciphertext *C* behaves to a large extent as a black box: all we can do with the box is keep it closed or opened in order to access and operate on the data.

In many situations this may be exactly what we want. For example, take a remote storage system, where we want to store a large collection of documents or data files. We store the data in encrypted form, and when we want to access a specific piece of data, we retrieve the corresponding ciphertext, decrypting it locally on our own trusted computer. But as soon as we go beyond the simple data storage/retrieval model, we are in trouble. Say we want the remote system to provide a more complex functionality, like a database system capable of indexing and searching our data, or answering complex relational or semistructured queries. Using standard encryption technology we are immediately faced with a dilemma: either we store our data unencrypted and reveal our precious or sensitive data to the storage/database service provider, or we encrypt it and make it impossible for the provider to operate on it.

If data is encrypted, then answering even a simple counting query (for example, the number of records or files that contain a certain keyword) would typically require downloading and decrypting the entire database content.

Homomorphic encryption is a special kind of encryption that allows operating on ciphertexts without decrypting them; in fact, without even knowing the decryption key. For example, given ciphertexts $C=Enc_K(M)$ and $C'=Enc_K(M')$, an additively homomorphic encryption scheme would allow to combine C and C' to obtain $Enc_K(M+M')$. Such encryption schemes are immensely useful in the design of complex cryptographic protocols. For example, an electronic voting scheme may collect encrypted votes $C_i=Enc_K(M_i)$ where each vote M_i is either 0 or 1, and then tally them to obtain the encryption of the outcome $C=Enc_K(M_1+..+M_n)$. This would be decrypted by an appropriate authority that has the decryption key and ability to announce the result, but the entire collection and tallying process would operate on encrypted data without the use of the secret key. (Of course, this is an oversimplified protocol, as many other issues must be addressed in a real election scheme, but it well illustrates the potential usefulness of homomorphic encryption.)

To date, all known homomorphic encryption schemes supported essentially only one basic operation, for example, addition. But the potential of fully homomorphic encryption (that is, homomorphic encryption supporting arbitrarily complex computations on ciphertexts) is clear. Think of encrypting your queries before you send them to your favorite search engine, and receive the encryption of the result without the search engine even knowing what the query was. Imagine running your most computationally intensive programs on your large datasets on a cluster of remote computers, as in a cloud computing environment, while keeping both your programs, data, and results encrypted and confidential. The idea of fully homomorphic encryption schemes was first proposed by Rivest, Adleman, and Dertouzos the late 1970s, but remained a mirage for three decades, the never-to-be-found Holy Grail of cryptography. At least until 2008, when Craig Gentry announced a new approach to the construction of fully homomorphic cryptosystems.

In the following paper, Gentry describes his innovative method for constructing fully homomorphic encryption schemes, the first credible solution to this long-standing major problem in cryptography and theoretical computer science at large. While much work is still to be done before fully homomorphic encryption can be used in practice, Gentry's work is clearly a landmark achievement. Before Gentry's discovery many members of the cryptography research community thought fully homomorphic encryption was impossible to achieve. Now, most cryptographers (me among them) are convinced the Holy Grail exists. In fact, there must be several of them, more or less efficient ones, all out there waiting to be discovered.

Gentry gives a very accessible and enjoyable description of his general method to achieve fully homomorphic encryption as well as a possible instantiation of his framework recently proposed by van Dijik, Gentry, Halevi, and Vaikuntanathan. He has taken great care to explain his technically complex results, some of which have their roots in lattice-based cryptography, using a metaphorical tale of a jeweler and her quest to keep her precious materials safe, while at the same time allowing her employees to work on them.

Gentry's homomorphic encryption work is truly worth a read. C

Daniele Micciancio is a professor in the computer science and engineering department at the University of California, San Diego.