A reprint from

American Scientist
the magazine of Sigma Xi, The Scientific Research Society

Alice and Bob in Cipherspace

Brian Hayes

Alice hands Bob a locked suitcase and asks him to count the money inside. "Sure," Bob says. "Give me the key." Alice shakes her head; she has known Bob for many years, but she's just not a trusting person. Bob lifts the suitcase to judge its weight, rocks it back and forth and listens as the contents shift inside; but all this reveals very little. "It can't be done," he says. "I can't count what I can't see."

Alice and Bob, fondly known as the first couple of cryptography, are really more interested in computational suitcases than physical ones. Suppose Alice gives Bob a securely encrypted computer file and asks him to sum a list of numbers she has put inside. Without the decryption key, this task also seems impossible. The encrypted file is just as opaque and impenetrable as the locked suitcase. "Can't be done," Bob concludes again.

But Bob is wrong. Because Alice has chosen a very special encryption scheme, Bob can carry out her request. He can compute with data he can't inspect. The numbers in the file remain encrypted at all times, so Bob cannot learn anything about them. Nevertheless, he can run computer programs on the encrypted data, performing operations such as summation. The output of the programs is also encrypted; Bob can't read it. But when he gives the results back to Alice, she can extract the answer with her decryption key.

The technique that makes this magic trick possible is called fully homomorphic encryption, or FHE. It's not exactly a new idea, but for many years it was viewed as a fantasy that would never come true. That changed in 2009, with a breakthrough discovery by Craig Gentry, who was then a graduate student at Stanford University. (He is now at IBM Research.) Since then, further refinements and more new ideas have been coming at a rapid pace.

A new form of encryption allows you to compute with data you cannot read

Homomorphic encryption is not quite ready for everyday use. The methods have been shown to work in principle, but they still impose a heavy penalty of inefficiency. If the system can be made more practical, however, there are applications ready and waiting for it. Many organizations are eager to outsource computation: Instead of maintaining their own hardware and software, they would like to run programs on servers "in the cloud," a phrase meant to suggest that physical location is unimportant. But letting sensitive data float around in the cloud raises concerns about security and privacy. Practical homomorphic encryption would address those worries, protecting the data against eavesdroppers and intruders and even hiding it from the operators of the cloud service.

Three's a Crowd

In the early days of their relationship, Alice and Bob kept no secrets from each other; it was the rest of the world they wanted to shut out. Their main problem was how to communicate privately over a public channel, where nosy third parties—such as Eve the eavesdropper—might be listening in. To solve this problem, Alice and Bob devised a variety of cryptographic schemes. Before sending a message to Bob, Alice would encrypt it with a secret key, turning plaintext into ciphertext; even if Eve intercepted the ciphertext, she could make no sense of it. But Bob had the decryption key, so he could recover the plaintext.

For some cryptosystems, Alice and Bob must each hold a copy of the same key, which both encrypts and decrypts. But then they face the thorny issue of how to transmit the key itself, without having it fall into Eve's hands. A particularly clever solution, called public-key cryptography, splits the key into two parts. Alice and Bob each publish a public encryption key, which allows anyone to send them an encrypted message. But they keep secret the corresponding decryption keys, so that only they can read the messages they receive.

Another innovation that helped Alice and Bob keep their private conversations out of the tabloid press was probabilistic cryptography, introduced in the early 1980s by Shafi Goldwasser and Silvio Micali of MIT. Earlier systems were deterministic: The same plaintext always produced the same ciphertext. But determinism is dangerous in public-key cryptography. Eve can try guessing the content of a message; then she encrypts the guess with the public key and checks to see if it matches an intercepted ciphertext. With a probabilistic scheme, every plaintext message has a multitude of possible encodings, and the system chooses randomly among them. Even if you correctly guess the plaintext, there's almost no chance of matching the random encryption. On decryption, however, all of the alternatives collapse to the same plaintext.

Cryptographic technology of this kind has become a routine part of life on the Internet—so routine that it often

Brian Hayes is senior writer for American Scientist. *Additional material related to the* Computing Science *column appears at http://bit-player.org. Address: 11 Chandler St. #2, Somerville, MA 02144. E-mail: brian@bit-player.org*

goes unnoticed. When you check your bank balance on the Web, or make an online purchase, you rely on a secure version of the hypertext transfer protocol (https rather than plain http), which provides a layer of encryption behind the scenes. Even Google searches are encrypted. These measures are meant to protect your messages while they are in transit. Encrypted communication shuts out Eve, who is sitting at the next table in Starbucks, tapping into your wifi connection. On the other hand, the cryptographic protocols conceal nothing from the recipients of your messages, who have the keys to decipher them. Usually, that's just fine, because the intended recipient is a trusted party. Homomorphic encryption is the tool for those occasions when you don't trust anyone, not even Bob.

A Parallel Universe

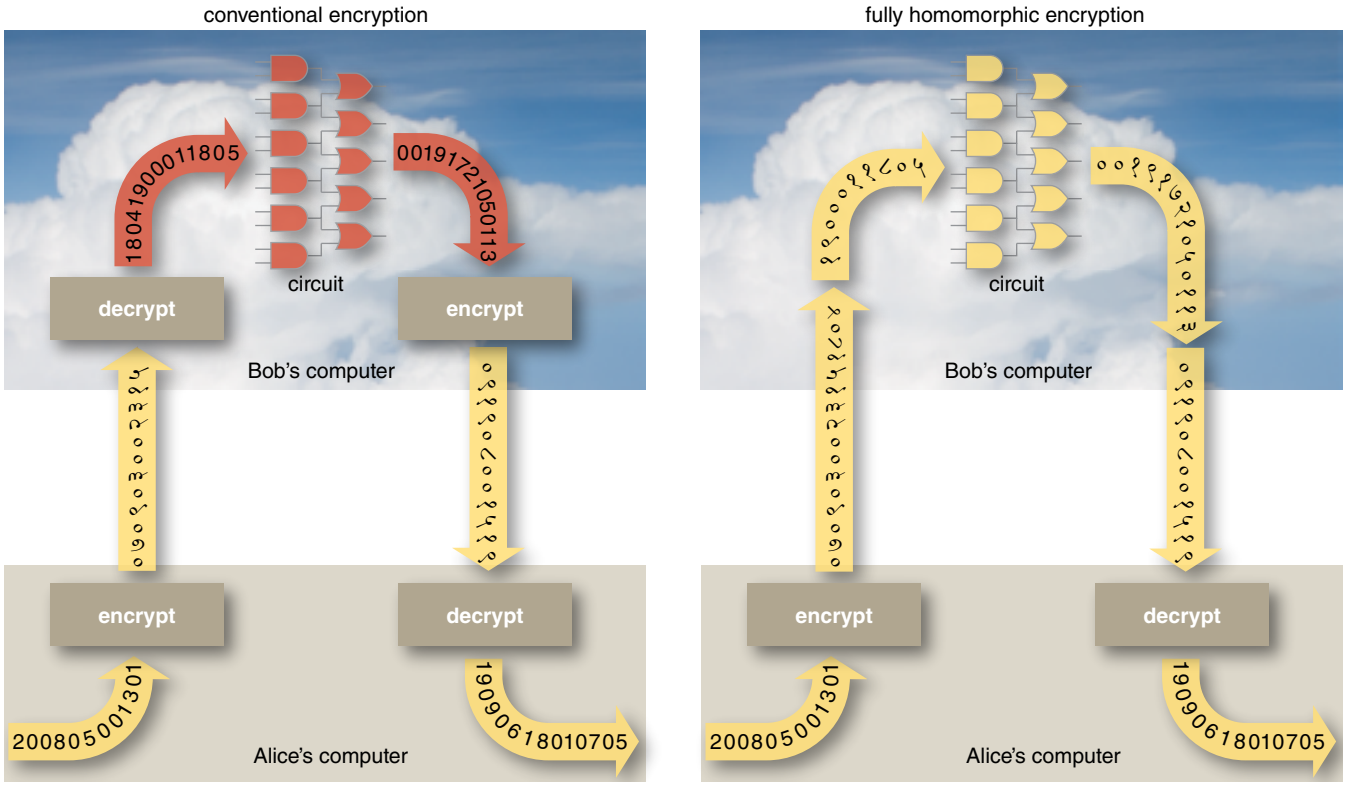
Over the years, Alice and Bob have gone their separate ways. Alice now works as the research director of a cryptographic software company; Bob has gone into hardware, running a cloud computing service. As they have drifted apart, their security and privacy needs have changed somewhat. When Alice talks to Bob, she still needs to guard against Eve's snooping. But, in addition, Alice's company now has proprietary information that she must not disclose to Bob. Complicating her predicament, she wants to use Bob's computers for tasks that involve the secret data.

Ordinary cryptography is no help in this situation. Alice can encrypt the data when she sends it to Bob, but he can do nothing with it unless he can decrypt it. That is exactly what Alice seeks to avoid. They are at an impasse, which homomorphic encryption is designed to surmount.

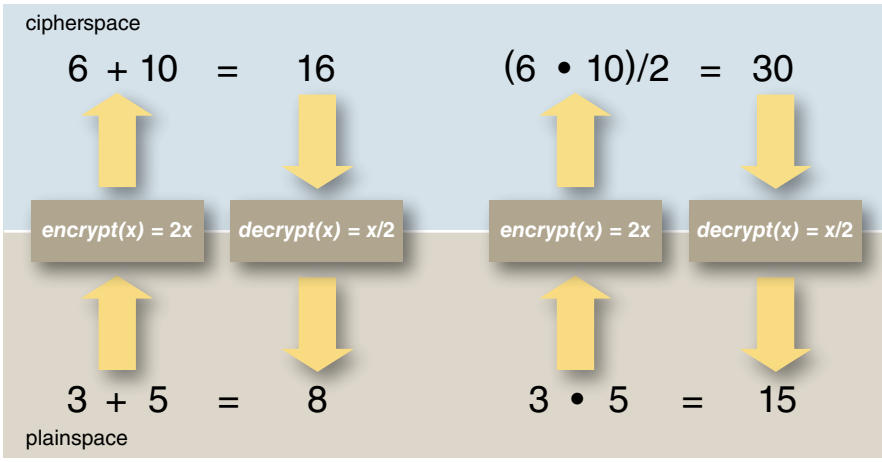
Before trying to explain how homomorphic encryption works, I should try to explain the word *homomorphic*. The Greek roots translate as *same shape* or *same form*, and the underlying idea is that of a transformation that has the same effect on two different sets of objects. The concept comes from the esoteric world of abstract algebra, but I can offer a more homely example, where the two sets of objects are the positive real numbers on the one hand and their logarithms on the other. Then multiplication of real numbers and addition of logarithms are homomorphic operations. For any positive real numbers x, y and z, if $x \cdot y = z$, then $\log(x) + \log(y) = \log(z)$. This homomorphism offers two alternative routes to the same destination. If we are given x and y, we can multiply them directly; or we can take their logarithms, then add, and finally take the antilog of the result. In either case, we wind up with z.

Homomorphic cryptography offers a similar pair of pathways. We can do arithmetic directly on the plaintext inputs x and y. Or we can encrypt x and y, apply a series of operations to the ciphertext values, then decrypt the result to arrive at the same final answer. The two routes pass through parallel universes: plainspace and cipherspace.

Arithmetic in plainspace is familiar to everyone. A number is conveniently represented as a sequence of bits (binary digits 0 and 1) and algorithms act



Alice has confidential data she wants to process on Bob's computer, which is a server "in the cloud." But she wants to make sure no one else gains access to the data—not even Bob. Conventional encryption *(left)* protects her information while it is in transit but not while the computation is underway on Bob's computer *(red portion of pathway)*. Homomorphic encryption *(right)* offers security from the moment the data stream leaves Alice's computer until it returns. The strategy requires that all the arithmetical and logical operations needed in the computation (symbolized here by a circuit of Boolean gates) be applied to the encrypted form of the data. In this diagram the distinction between encrypted and unencrypted data—between ciphertext and plaintext—is suggested by a typographic convention: The ciphertext is shown in numerals of the Devanagari alphabet, which have the sequence ०१२३४५६७८९.



The concept of homomorphism describes a parallel linkage between operations on two sets of objects. In this toy example the sets of objects are the set of all integers *(lower panel)* and the set of even integers *(upper panel)*. The operations on the objects are addition and multiplication. Going back and forth between the two sets is just a matter of doubling or halving a number. Addition works the same way in both sets. In the case of multiplication, an adjustment is needed: For even numbers, the product of x and y is defined as $(x \cdot y)/2$. These sets and operations can be pressed into service as a rudimentary homomorphic cryptosystem. Plaintext integers are encrypted by doubling; then any sequence of additions and multiplications can be carried out; finally the result is decrypted by halving.

on the bits according to rules of logic and arithmetic. Among the many operations on numbers we might consider, it turns out that adding and multiplying are all we really need to do; other computations can be expressed in terms of these primitives.

Doing mathematics in cipherspace is much stranger. Indeed, the task seems all but impossible. Encryption is a process that thoroughly scrambles the bits of a number, whereas algorithms for arithmetic are extremely finicky and give correct results only if all the bits are in the right places. Nevertheless, it *can* be done.

As a proof of concept, I offer an extremely simple homomorphic cryptosystem. Assume the plaintext consists of integers. To encrypt a number, double it; to decrypt, divide by 2. With this scheme we can do addition on enciphered data as well as a slightly nonstandard version of multiplication. Given plaintext inputs x and y, we can encrypt each of them separately, add the ciphertexts, then decrypt the result. This roundabout calculation gives the correct answer because $2x+2y = 2(x+y)$.

To make multiplication come out right, we have to define the product of ciphertexts as $(x \cdot y)/2$, whereas plaintexts are multiplied by the usual formula $x \cdot y$. With this rule it's easy to verify that the three-step sequence *encrypt-multiply-decrypt* yields the same result as simply multiplying the plaintexts. (Fiddling with definitions in order to get the right answer may seem like cheating, but many mathematical objects come with their own idiosyncratic rules for multiplication. Two examples are matrices and complex numbers.)

As cryptosystems go, the doubling scheme is certainly simple, and it's fully homomorphic. We can do all the arithmetic we want on ciphertexts. On the other hand, the system is not recommended if you actually want to keep secrets. Doubling a number does *not* thoroughly scramble the bits; it merely shifts them left by one position.

Devising a *secure* fully homomorphic cryptosystem is much harder. That's what Gentry accomplished in 2009. Making the system efficient enough for practical applications is yet another challenge, still being addressed.

First Date

The idea of computing with encrypted data was first proposed in 1978 by Ron Rivest, Len Adleman and Michael L. Dertouzos, who were all then at MIT. Just a few months before, Rivest and Adleman, along with Adi Shamir, had introduced the first implementation of a public-key cryptosystem, which came to be known as RSA after their initials. (The RSA paper, by the way, also introduced Alice and Bob in their debut performance as celebrity cryptographers.)

The basic RSA scheme is partially homomorphic: It allows multiplication of ciphertexts but not addition. Rivest, Adleman and Dertouzos pointed out this fact and also mentioned a few other ways to achieve partial "privacy homomorphisms." They asked whether it would be possible to construct a secure scheme capable of general computation on ciphertexts.

In the next 30 years there were occasional advances on this front. For example, in 2005 Dan Boneh, Eu-Jin Goh and Kobbi Nissim devised a homomorphic system that allowed an unlimited number of additions on the ciphertext, followed by a single multiplication. (Boneh, by the way, was Gentry's thesis advisor.) In spite of such incremental progress, however, Gentry's announcement of a fully homomorphic scheme came as a total surprise in 2009.

Noisy Arithmetic

In broad outline, here is Gentry's FHE construction kit. He creates a cryptosystem with the usual *encrypt* and *decrypt* functions, which convert bits from plaintext to ciphertext and back. He also builds an *evaluate* function that accepts a description of a computation to be performed on the ciphertext. The computation is specified not as a sequential program but as a circuit or network, where input signals pass through a cascade of logic gates. Such circuits are most often assembled from Boolean gates (AND, OR, NOT, etc.), but they can also be specified in terms of addition and multiplication steps.

The *evaluate* function amounts to a complete computer embedded in the cryptosystem. In principle, it can calculate any computable function, provided that the circuit representing the function is allowed to extend to arbitrary depth. The depth of a circuit is the number of gates on the longest path from input to output. A full-powered computer must be able to handle circuits of arbitrary depth. Here the homomorphic system runs into a barrier. The problem is that ciphertext data are contaminated with numerical "noise"—slight discrepancies from their ideal values. Every arithmetic operation amplifies the noise, until eventually it overwhelms the signal.

The origin of the noise lies in the probabilistic encryption process. Think of each ciphertext value as a point in

space. The probabilistic *encrypt* function injects a smidgen of randomness into each of the point's coordinates, displacing it slightly from the position it would occupy in a deterministic cryptosystem. The *decrypt* function filters out the noise by treating each point as if it were located at the nearest unperturbed position. When the noise is amplified by homomorphic computations, however, the point wanders farther from its correct position, until finally the *decrypt* function will associate it with an incorrect plaintext value.

Roughly speaking, each homomorphic addition doubles the noise, and each multiplication squares it. Hence the number of operations must be limited or errors will accumulate. Because of the limit on circuit depth, this version of the cryptosystem cannot be called fully homomorphic but only "somewhat homomorphic."

The depth limit could be evaded in the following way: Whenever the noise begins to approach the critical threshold, decrypt the data and then re-encrypt it, thereby resetting the noise to its original low level. The trouble is, decryption requires the secret key, and the whole point of FHE is to allow computation in a context where that key is unavailable.

The Pause That Refreshes

This is where the story gets wacky and wonderful. The *evaluate* function built into the cryptosystem is capable of performing any computation, provided it does not exceed the noise limit on circuit depth. So we can ask *evaluate* to run the *decrypt* function. *Evaluate* is designed to work with encrypted data, so the secret key supplied to it in this circumstance is an encrypted version of the normal key; specifically, the secret key supplied to *decrypt* running within *evaluate* is the ciphertext produced when *encrypt* is applied to the plaintext of the secret key. When *decrypt* is run with this enciphered key, the result is not plaintext but a new encryption of the ciphertext, with reduced noise.

In effect, Alice is giving Bob a copy of the key needed to unlock the data, but the key is inside a securely locked box and can only be used within that box. As a matter of fact, the box is locked with the very key that is locked inside the box! (Gentry discusses an even more elaborate version of this dizzying metaphor, in which Alice manufactures jewelry in the locked boxes.)

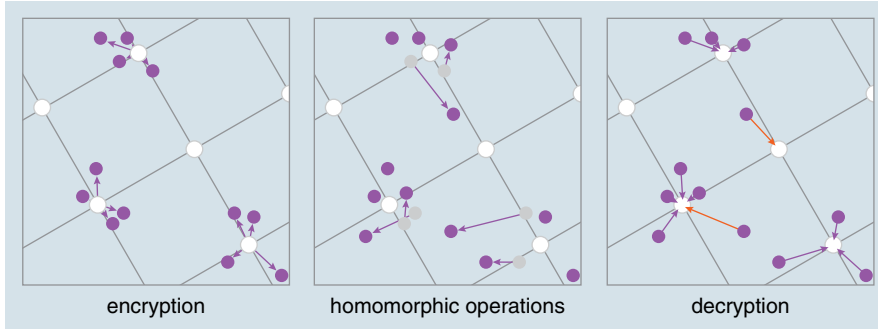
The pause to re-encrypt and refresh the noisy ciphertext can be repeated as needed. In this way the computer can handle a circuit of any finite depth, and the system becomes fully homomorphic. It can carry out arbitrarily complex computations on encrypted data.

An essential assumption in this scheme is that the *decrypt* circuit is itself shallow enough to run without exceeding the noise threshold. Indeed, its depth needs to be a little less than the limit, or else the computer will spend all its time refreshing the data and will never accomplish any useful work. When Gentry first formulated his FHE scheme, he found that this condition was not met. The *evaluate* function could not run the *decrypt* routine without accumulating excessive noise. The remedy was a technique for "squashing" *decrypt*, at the cost of making the key larger and more complicated. With this last innovation, the problem was solved.

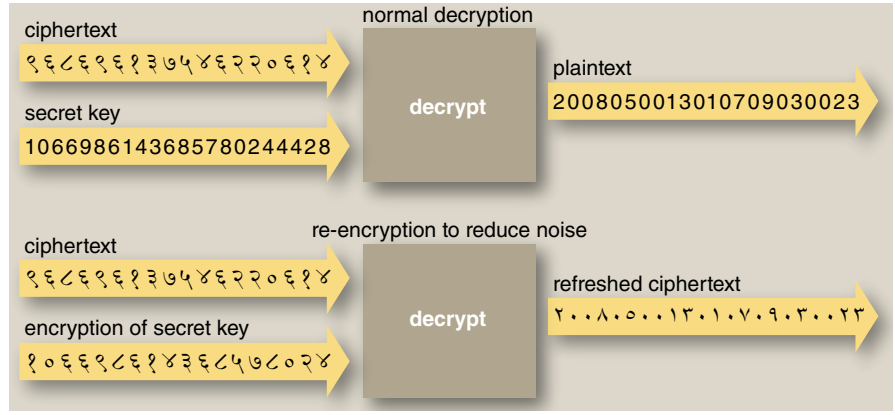
Hard Problems

Gentry described his FHE system in his doctoral dissertation and in a paper at the Symposium on the Theory of Computing in 2009. In the three years since then, dozens of variations, elaborations and alternative schemes have been published, along with at least three attempts to implement homomorphic encryption in a working computer program.

Most of the systems share the same overall architecture, with a somewhat homomorphic scheme that gets promoted to full homomorphism. Where the ideas differ is in the underlying cryptographic mechanism—the way that bits are twiddled and secrecy is achieved.



Random "noise" in a secure cryptosystem is the principal impediment to homomorphic operation. Encrypted data can be envisioned as points *(purple disks)* that are given small random displacements from a finite set of lattice points *(white disks)*. On decryption, each purple disk is attracted to the nearest white lattice point. Homomorphic operations amplify the random displacements. If the noise level exceeds a threshold, some of the disks gravitate to the wrong lattice point, leading to an incorrect decryption *(red arrows)*. Without some means of noise control, the system can support only a limited number of homomorphic operations.



A noise-abatement mechanism was invented by Craig Gentry in 2009. Gentry observed that if a noisy ciphertext could be decrypted and then re-encrypted, it would be "refreshed," with reduced noise. But decryption requires a secret key, which is not available. The solution is to run the ciphertext through the decryption algorithm, but with an encrypted version of the decryption key. The result is a new ciphertext, just as secure as the original but with lower noise. (Here the refreshed ciphertext is represented by numerals from another alphabet, this one Arabic.)

Every cryptosystem is based on a problem that's believed to be hard in general (so that Eve can't solve it) but easy if you know a shortcut (so that Alice and Bob can decrypt messages efficiently). RSA's hard problem is the factoring of large integers; the shortcut is knowledge of the factors. Gentry's 2009 algorithm relies on a problem from the theory of integer lattices—sets of discrete points arranged like the atoms of a crystal in a high-dimensional space. Lattices give rise to an abundance of computationally difficult problems. For example, from a random position in space it is hard to find the closest lattice point unless you happen to know a specific set of coordinates that serve as a geometric guidebook to the lattice.

In 2010 another homomorphic cryptosystem was invented by Marten van Dijk of MIT, Gentry, Shai Halevi of IBM and Vinod Vaikuntanathan, now at the University of Toronto. In this case the hard problem comes from number theory; it's called approximate GCD. The exact GCD, or greatest common divisor, is easy to calculate; Euclid gave an efficient (and famous) algorithm. A "noisy" version of the problem seems to be much harder. If two large numbers have the GCD p, and you alter those numbers by adding or subtracting small random quantities, it becomes difficult to find p. In the cryptosystem, p is the secret key.

A problem called learning with errors forms the basis of a third FHE system introduced by Zvika Brakerski of Stanford and Vaikuntanathan. Here the task is to solve a system of simultaneous equations where each equation has some small probability of being false. As with GCD, this is an easy problem in the exact case, where there are no errors, but searching for a subset of correct equations is laborious.

More recently, Brakerski, Vaikuntanathan and Gentry have developed a variant of the learning-with-errors system that takes a different approach to noise management. Instead of stopping the computation at intervals to re-encrypt the data, they incrementally adjust parameters of the system after every computational step in a way that prevents the noise level from ever approaching the limit.

Working Code

Computing in cipherspace is a cute theoretical novelty, but can it ever become a practical technology? Questions of computational efficiency and overhead are more challenging in FHE than in other kinds of cryptography. When encryption is used only to create a secure communications channel, it has no direct effect on the efficiency of computations done at either end of the connection. Homomorphic encryption is different: The cryptosystem becomes the computing platform, and any inefficiency slows the entire process.

Many homomorphic schemes exact a high price for security. During encryption, data undergo a kind of cosmic inflation: A single bit of plaintext may blow up to become thousands or even millions of bits of ciphertext. The encryption key can also become huge—from megabytes to gigabytes. Merely transmitting such bulky items would be costly; computing with the inflated ciphertext makes matters worse. Whereas adding or multiplying a few bits of plaintext can be done with a single machine instruction, performing the same operation on the inflated ciphertext requires elaborate software for high-precision arithmetic.

Much current work is directed toward mitigating these problems. For example, instead of encrypting each plaintext bit separately, multiple bits can be packed together, thereby "amortizing" the encryption effort and reducing overhead.

The ultimate test of practicality is to create a working implementation. Nigel P. Smart of the University of Bristol and Frederik Vercauteren of the Catholic University of Leuven were the first to try this. They built a somewhat homomorphic system, but could not extend it to full homomorphism; the bottleneck was an unwieldy process for generating huge encryption keys.

Gentry and Halevi, working with a somewhat different variant of the lattice-based algorithm, did manage to get a full system running. And they didn't need to build it on IBM's Blue Gene supercomputer, as they had initially planned; a desktop workstation was adequate. Nevertheless, the public key ballooned to 2.3 gigabytes, and generating it took two hours. The noise-abating re-encryptions took 30 minutes each.

In another implementation effort, Kristin Lauter of Microsoft Research, Michael Naehrig of the Eindhoven Institute of Technology and Vaikuntanathan show that large gains in efficiency are possible if you are willing to compromise on the requirement of full homomorphism. They do not promise to evaluate circuits of unbounded depth, but instead commit only to some small, fixed number of multiplications, along with unlimited additions. They have working code based on the learning-with-errors paradigm. Except at the highest security levels, key sizes are roughly a megabyte. Homomorphic addition takes milliseconds, multiplication generally less than a second. These timings are a vast improvement over earlier efforts, but it's sobering to reflect that they are still an order of magnitude slower than the performance of the ENIAC in 1946.

Putting Code to Work

Lauter, Naehrig and Vaikuntanathan also discuss some of the ways we might use homomorphic computing. Ensuring the privacy of online medical records is one application. The patient would grant doctors access to selected records by sharing a secret key.

Wall Street is another potential customer for homomorphic services. The "quants" who base investment decisions on computational analysis have a strong proprietary interest not only in their data but also in their algorithms. With FHE both can be protected by the same mechanism.

A third idea is to build a cryptographic privacy fence between online advertisers and consumers. Advertisers, eager to reach individuals with specific interests or habits, gather and cross-index data on people's activities on the Internet and elsewhere. A service based on homomorphic encryption could match ads to targeted consumers while ensuring that advertisers learn nothing about the people selected.

When I asked Vaikuntanathan what application he thought might be deployed first, he had another suggestion: spam filtering. If you publish a public key and invite correspondents to send you encrypted email, a spammer can take advantage of the key to encrypt advertisements and the other effluvia that fills our mailboxes. Spam-filtering services cannot read and reject the encrypted spam unless you are willing to share your decryption key; homomorphic encryption could solve that problem.

My own fantasy application is an offshore bank called the Homomorphic Trust Company. The online interface might look much the same as any other bank's, with the usual cryptograph-

ic safeguards against intruders. But at this bank, even the bankers could not know the details of your transactions. I think Alice might be interested; she could get rid of that suitcase full of uncountable cash.

Bibliography

Boneh, D., E.-J. Goh and K. Nissim. 2005. Evaluating 2-DNF formulas on ciphertexts. In *Proceedings of Theory of Cryptography*, pp. 325–341.

Brakerski, Z., C. Gentry and V. Vaikuntanathan. 2011. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the Third Innovations in Theoretical Computer Science Conference*, pp. 309–325.

Brakerski, Z., and V. Vaikuntanathan. 2011. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Proceedings of the 31st Annual Cryptology Conference*, pp. 505–524.

Coron, J., D. Naccache and M. Tibouchi. 2011. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Proceedings of Eurocrypt 2012*, pp. 446–464.

Gentry, C. 2009. A fully homomorphic encryption scheme. Ph.D. dissertation. Stanford University. Available at http://crypto.stanford.edu/craig.

Gentry, C. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st ACM Symposium on Theory of Computing*, pp. 169–178.

Gentry, C. 2010. Computing arbitrary functions of encrypted data. *Communications of the ACM* 53(3):97–105.

Gentry, C., and S. Halevi. 2011. Implementing Gentry's fully-homomorphic encryption scheme. In *Proceedings of Eurocrypt 2011*, pp. 129–148.

Gentry, C., S. Halevi and N. P. Smart. 2012. Fully homomorphic encryption with polylog overhead. In *Proceedings of Eurocrypt 2012*, pp. 465–482.

Goldwasser, S., and S. Micali. 1982. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pp. 365–377.

Gordon, J. 1984. The story of Alice and Bob. http://www.johngordonsweb.co.uk/concept/alicebob.html.

Lauter, K., M. Naehrig and V. Vaikuntanathan. 2011. Can homomorphic encryption be practical? In *Proceedings of the Third ACM Workshop on Cloud Computing Security*, pp. 113–124.

Rivest, R. L., L. Adleman and M. L. Dertouzos. 1978. On data banks and privacy homomorphisms. In *Foundations of Secure Computation* (New York: Academic Press), pp. 169–180.

Smart, N. P., and F. Vercauteren. 2010. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Proceedings of the Conference on Practice and Theory in Public Key Cryptography*, pp. 420–443.

van Dijk, M., C. Gentry, S. Halevi and V. Vaikuntanathan. 2010. Fully homomorphic encryption over the integers. In *Proceedings of Eurocrypt 2009*, pp. 24–43.