



Image © Alessia Pierdomenico/Reuters/Corbis

British Turing Bombe machine at Bletchley Park Museum, Bletchley, United Kingdom

Breaking the Code

The Academy's 1969th Stated Meeting featured members of the Catalyst Collaborative@MIT performing a staged reading of Hugh Whitmore's play *Breaking the Code*, about the life of Alan Turing. (See opposite page for an extract from the play.) The reading was followed by a panel discussion of Turing's professional and personal life from the perspective of science, engineering, drama, and social history. Academy President Leslie C. Berlowitz, who welcomed Fellows and guests to the program, remarked that Turing "was instrumental in breaking the Nazis' Enigma code during World War II. Yet his own life was a cipher. He could not resolve the tension between his sexual orientation and prevailing government policies and public mores. The play portrays these conflicts and how they tormented his life."

Academy Fellow Alan Lightman (MIT), a cofounder of the Catalyst Collaborative@MIT, described the group's mission "to use theater to convey the culture of science to the public, with an emphasis on subjects of social and cultural interest." He also stressed that "the hallmark of their productions is the post-performance discussion during which the audience can ask questions of Boston-area scientists whom we have invited to the performance."

Following the staged reading, Academy Fellows Laurence Senelick (Tufts University), Ronald L. Rivest, Shafi Goldwasser, and Silvio Micali (all from MIT) described the impact of Turing's work and life on cryptology and computer and network security, as well as how it is portrayed in *Breaking the Code*.

An edited transcript of the panel discussion follows on pages 24 to 30.

From the play *Breaking the Code*, by Hugh Whitemore

Act II, Scene 7

TURING. Thank you, Nikos, dear. Thank you. It's a good feeling, isn't it? Solving a problem, finding the answer. Making it work. A good feeling. It's all like that wireless, really, it's all a question of making the right connections. Shall I tell you a secret? Top secret. I couldn't even tell my analyst about this. But since you won't understand a single word, it doesn't really matter. It all took place at the beginning of the war in a country house called Bletchley Park. The Germans had built a machine called the Enigma. It was very cunning. It made codes – and nobody knew how to break the codes it made. That was the problem we had to solve. If we didn't, if we couldn't we'd lose the war – it was as simple as that. But where to begin? Well, first there was guess-work. The codebreaking process always began with a guess. You had to guess what the first few letters of the message might mean. This wasn't as difficult as it sounds because military messages invariably start with a stereotyped phrase: The date, the time, the name and rank of the sender, that sort of thing. Then we discovered that it was possible to use the phrase we guessed to form a chain of implications, of logical deductions, for each of the rotor positions. If this chain of implications led you to a contradiction – which was usually the case – that meant you were wrong, and you'd have to move onto the next position. And so on and so on. An impossibly lengthy and laborious process; time was against us. We didn't know what to do. And then, one afternoon, I remembered a conversation I'd had with Wittgenstein; we were arguing about the fact that a contra-

diction implied any proposition. And I saw – immediately – how I could use this elementary theorem in mathematical logic to build a machine that would have the necessary speed: a machine with electrical relays and logical circuits which would sense contradictions and recognize consistencies; a machine of cribs, closed loops, and perfect synchrony; a machine for discerning a pattern in the patternless. If your guess was wrong, then the electricity would flow through all the related hypotheses and knock them out in a flash – like the chain reaction in an atomic bomb. If your guess was correct, everything would be consistent – and the electrical current would stop at the correct combination. Our machine could examine thousands of millions of permutations at amazing speed – and, with any luck, would give us the “way in.” More than that: all the connections had been made. There was the pure beauty of the logical pattern. The human element. The deeply satisfying relationship between the theoretical and the practical. What a moment that was. Quite, quite extraordinary. Oh, Christopher . . . If only you could've been there. Never again. Never again a moment like that. In the long run, it's not breaking the code that matters – it's where you go from there. That's the real problem.

Breaking the Code by Hugh Whitemore, based on the book *Alan Turing: The Enigma* by Andrew Hodges. *Breaking the Code* will be produced again on Broadway in 2012. Look to the theater announcements!



Alan Lightman

Alan Lightman is Adjunct Professor of Humanities at the Massachusetts Institute of Technology. He was elected a Fellow of the American Academy in 1996.

The Catalyst Collaborative – formed five years ago by Debra Wise, artistic director of the Underground Railway Theater, a Boston theater company; Alan Brody, a playwright and professor of theater at MIT; John Lipsky, a playwright and director at Boston

paintings, and other art forms. Artists have also found that the general public has a real hunger for science and that many in the public don't want to encounter science through the traditional channels.

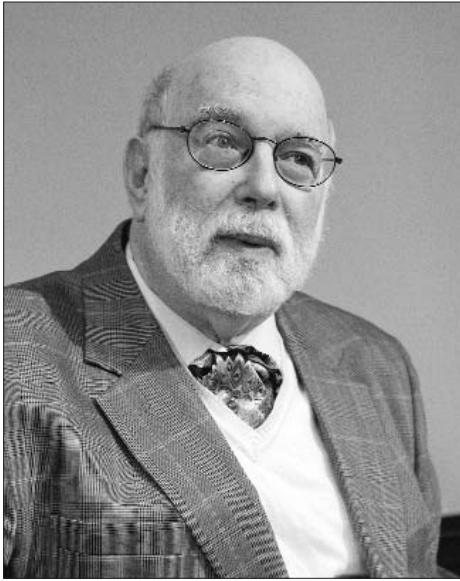
The Catalyst Collaborative does several things. Our mission is to create new plays about science. We bring scientists and playwrights together at an early stage to create the play. We also produce plays, including, recently, *An Evening with Richard Feynman*, *The Life of Galileo*, *Einstein's Dreams*, *The Water Engine* by David Mamet, and a play we commissioned, *From Orchids to Octopi* by Melinda Lopez. We usually perform in the Central Square Theater but have performed all over the Boston/Cambridge area. We like to bring in a large range of the community. One of the hallmarks of our productions is the post-performance discussion during which the audience can ask questions of Boston-area scientists whom we have invited to the performance.

[At this point, there is a staged reading of *Breaking the Code*.]

The general public has a real hunger for science and many in the public don't want to encounter science through the traditional channels.

University; and myself – is an imaginative collaboration between MIT and the Underground Railway Theater. The Collaborative's mission is to use theater to convey the culture of science to the public, with an emphasis on subjects of social and cultural interest.

Over the last quarter of a century, we have seen a surge of theater involving science, including such outstanding examples as Tom Stoppard's *Arcadia* (1993), Michael Frayn's *Copenhagen* (1998), and David Auburn's *Proof* (2000). Artists have long been interested in science as a source of subject matter – not just for plays but for novels,



Laurence Senelick

Laurence Senelick is Fletcher Professor of Drama and Oratory and Director of Graduate Studies at Tufts University. He was elected a Fellow of the American Academy in 2010.

As Alan Lightman pointed out, one of the most interesting phenomena in drama in the last twenty years or so has been the number of plays that have been organized around scientific ideas or around the lives of scientists. *Arcadia*, for instance, treats the origins of chaos theory and iteration. The human relationships in *Proof* pivot on a paradigm-shifting proof about prime numbers. In *Copenhagen*, Werner Heisenberg and Niels Bohr argue over the ethics and control of nuclear power. Audiences leave these performances feeling that they have been edified as well as entertained.

Breaking the Code, because of its biographical thrust, more closely resembles the granddaddy of these plays: Bertolt Brecht's *Das Leben des Galilei*, *The Life of Galileo*. For Brecht, Galileo was the perfect example of the intellectual in conflict with the powers that be.

In the first version of Brecht's play in 1937, Galileo is a sort of wily antifascist who manages to get his ideas across despite a repressive power structure. When Brecht rewrote the play in 1945, after the A-bomb had been dropped, Galileo became quite a different

figure. In this version, Galileo is complicit with power in some rather tricky moral issues. He no longer sees the establishment as simply an antagonist but goes along with it for his own reasons. This is obviously a more interesting and complex treatment of the situation.

Breaking the Code presents a man, Alan Turing, who believes in the power of the machine to solve problems. And yet what keeps tripping him up is his humanity, particularly what would have been regarded as a great rift running through that humanity: his sexuality.

Turing picked a particularly unfortunate time to be a homosexual. During World War II, both in England and America, in civilian life and in the military, rules on sexual activ-

call gay men (the term was used only by co-teries at the time). Clubs and public toilets were raided on a regular basis. Individuals who would normally have been above suspicion suddenly were being investigated and made the centers of high-profile prosecutions. In the United States, Senator McCarthy equated Reds and fags. The State Department and the Pentagon purged many inoffensive people simply on the basis of suspicion.

Turing, who was by no means a flamboyant individual, thus had to live what we might call a coded life. *Breaking the Code* deals not only with his experiments at Bletchley Park, where the Enigma code was broken, but also with the fact that to be a homosexual in that society meant learning a cipher

Breaking the Code presents a man who believes in the power of the machine to solve problems. And yet what keeps tripping him up is his humanity, particularly his sexuality.

ity were relaxed. With the threat of death hanging over everybody's head, sex was something to be revealed in, to be welcomed and entertained whenever it showed up, because who knew the next time one might have the opportunity. And so a good deal of activity that would ordinarily have been policed was winked at.

Things changed radically after the war. England became particularly censorious because of the Cambridge Five, a group of men allegedly recruited at Cambridge University to spy for the Soviet secret service. Two of them, Guy Burgess and Donald Maclean, left England for the Soviet Union in 1951 after being tipped off about a British intelligence service investigation into their activities. Only the authorities knew of their defection until the two men appeared at a press conference in Moscow in 1956.

The open secret that Burgess was homosexual spurred a witch hunt of what we now

that you would then use and interpret at your own risk.

For instance, Polari, based on the Romany language and current in show business, was also exploited by the homosexual subculture. By dropping certain Polari words and phrases – as well as other hints, including sartorial signals – you could both communicate your trustworthiness and establish whether you were dealing with somebody you could trust, somebody who was a sexual fellow traveler.

Throughout the play, Turing is looking for that trust. He is not looking so much for the moment of passion, although the moment of passion, the one-night stand, is clearly a way of gauging the relationship. Instead, he is looking for a situation where he can be as comfortable with a human being as he is with a machine. Often he is looking for predictability, for something that will be consistent, that will always provide the right

answer in a given case. That is all very well and good when you are dealing with mathematical formulas; it is much more difficult when you are dealing with human behavior.

Turing's fate was ridiculous by any law of equity or logic. He took home a man whose friend then broke into Turing's apartment and stole a few things. Turing foolishly informed the police, who put more than two and two together. Turing was arrested and prosecuted under the same law that had condemned Oscar Wilde a half-century earlier: the 1885 Labouchere Amendment, which proscribed all sexual activity between two males, whether private or public.

Upon his conviction, Turing was given the choice of a prison sentence or chemical castration by hormone injection. He chose the latter. His security clearance was revoked, and he was not allowed into the United States. All the things on which he had predicated his life, all the things that had enabled him to function as a human being, were suddenly pulled out from under him. He died of cyanide poisoning, ruled a suicide, two years later.

Turing structured the professional achievements of his life on consistency, on mechanical rectitude, but he was continually being tripped up by the fallacies of human behavior and human sexuality. His story is both farcical and tragical. It is that ambiguity that makes the play's portrayal of Turing so engrossing.



Ronald L. Rivest

Ronald L. Rivest is Andrew and Erna Viterbi Professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. He was elected a Fellow of the American Academy in 1993.

Arguably, the breaking of the German Enigma code was the most significant event in the history of cryptography; it marked a dramatic, qualitative change. Before World War II, cryptography felt like a

legitimized the deep study of mathematics and computation for warfare.

The use of cryptanalysis in World War II may not have changed the outcome of the war. But certainly it shortened the war considerably by allowing the Allies to break codes describing where German U-boats were going to attack.

Cryptology is all about the study of secrets – about generating secrets and uncovering secrets – which is why this play about Turing is so marvelous; it deals with secrets on a number of different fronts. If you find out someone's secret, you have an advantage. If they find out your secret, they have an advantage. If you can guard your secrets, you can protect yourself. Cryptography is all about the mathematical generation and use of secrets. You identify yourself because of the secrets you own. If you have a digital certificate in your browser, that's a secret you use to identify yourself.

Once you know someone's secret, a second layer of secrecy emerges as you try to keep hidden the fact that you have found out somebody else's secret. During World War II, Winston Churchill faced the issue of what to do with secretly obtained information.

The brilliant work done at Bletchley was founded on the ability to build machines that used not only the cutting-edge technology of the day to run through the code's possibilities but also used clever algorithms and clever mathematics to reduce the number of possibilities to a manageable level.

field for amateurs, dabblers. After World War II, it became serious business because of the impact of Turing's work and the work of his colleagues at Bletchley Park. Breaking the Enigma code was to mathematicians what the atomic bomb was to physicists; it

Because the British had broken the Enigma code, they were able to learn of the German plans to bomb Coventry. But if Churchill warned of the impending bombing, he might also reveal the secret that the Brits knew how to break the German code.

What was the impact of Turing's cryptanalysis work at Bletchley? (Cryptanalysis is the breaking of codes. Cryptography is the making of codes, although the word *cryptography* has come to refer to both practices.) Turing was a cryptanalyst focused on breaking the German Enigma code.

Turing had an impact on the field, but not in the usual academic sense. He didn't write papers in this area (at least I don't know of any cryptography papers he wrote that are unclassified). He was not trying to build a science or create a field of academic inquiry. He was trying to solve the problem at hand, to do whatever it took to break that code.

His impact on academic cryptography was indirect and primarily through Claude Shannon, whom Turing met several times in 1943 at Bell Labs in New Jersey to discuss the breaking of Enigma. After the war Shannon wrote a seminal academic paper about cryptography.

Turing's work thus became influential through the publications of others but not because these publications revealed the technical details of breaking Enigma, some parts of which are still hidden. Instead, the powerful idea of *building machines to solve problems* pervades his work, both theoretical and practical. The brilliant work done at Bletchley was founded on the ability to build machines that used not only the cutting-edge technology of the day to run through the code's possibilities but also used clever algorithms and clever mathematics to reduce the number of possibilities to a manageable level.

Turing also designed machines for theoretical purposes. The eponymous machine he designed in the 1930s still plays a role today in the foundations of theoretical computer science.

The work done at Bletchley Park further laid the foundation for computer science in practice. Many of the earliest computers were spin-offs of work done at Bletchley. People who learned how to build interesting electronic devices that could do computa-

tion of various sorts at Bletchley Park later went off to build the first generation of digital computers. Turing is thus the father of both the practice and theory of computer science.

The work done at Bletchley also became a model for intelligence agencies everywhere. The U.S. National Security Agency, founded in the 1950s, can be viewed as a scaled-up version of Bletchley Park: many smart mathematicians in one place with many high-powered computers, doing their best to break the codes of the day.

The field of cryptography has grown substantially since the end of World War II; it has become an academic discipline and has expanded to include nonmilitary applications. Cryptography is the glue that holds together our electronic commerce, the glue that makes the Web work effectively so you know to whom you are talking. My colleagues Shafi Goldwasser and Silvio Micali are pioneers in the field, having helped lay the foundations of public key cryptography and many of the other wonderful cryptographic advances since the war.

But the idea of building machines to solve our problems, an idea that characterizes our information age society, owes much to Turing's pioneering efforts, both theoretical and practical.



Shafi Goldwasser

Shafi Goldwasser is RSA Professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology and Professor of Computer Science and Applied Mathematics at Weizmann Institute of Science in Israel. She was elected a Fellow of the American Academy in 2001.

The play *Breaking the Code* describes Alan Turing as a mathematician, albeit one who is doing a very unusual kind of mathematics. Were he alive today, he probably would not be called a mathematician. He would be called a theoretical computer scientist, which is appropriate because he was the first computer scientist – even though he never built or programmed a real computer.

What is the difference between mathematics and computer science? Mathematics is the study of what is true; it provides a framework of rules and qualifying mathematical facts. For example, what is the function that defines the circumference of a circle, or what is the product of number x and number y . Computer science is the study of how to compute the circumference of a circle, how to multiply two numbers. Computer science does not ask what is the product of x and y but how do we actually find it?

The central notion in computer science, then, is the notion of computing, of a com-

putational process or an algorithm that describes a computational process. Wikipedia defines computational process, or algorithm, as an effective method by which to express a finite list of well-defined instructions for calculating a function.

An algorithm or a computational process is a sequence of well-defined steps. But what does that mean? What is an effective method? What are these well-defined steps? Turing, motivated by what is called a *decision problem* and by his work with machines during the war, set out to answer this question. That is, he tried to capture an effective method of computation in a precise and accurate mathematical way.

What he came up with is a definition of a machine that today is called a Turing ma-

chine. The machine is simple, comprising three parts. First, the machine has an infinite tape (imagine an infinite roll of paper) divided into squares. Second, the machine has a head, a pointer into the tape. The head can either read symbols from the tape or write symbols on the tape. Third, the machine has an automaton, a gearbox. After a symbol on the tape has been read, the gearbox looks at the content of what was read and decides whether to shift gears and whether to write something else on the tape.

Turing, motivated by what is called a decision problem, tried to capture an effective method of computation in a precise and accurate mathematical way.

chine. The machine is simple, comprising three parts. First, the machine has an infinite tape (imagine an infinite roll of paper) divided into squares. Second, the machine has a head, a pointer into the tape. The head can either read symbols from the tape or write symbols on the tape. Third, the machine has an automaton, a gearbox. After a symbol on the tape has been read, the gearbox looks at the content of what was read and decides whether to shift gears and whether to write something else on the tape.

Computing the circumference of a circle or multiplying two numbers will require a relatively small number of instructions. They would be carried out in record speed on any of today's computers or calculators. But on the machine Turing defined, the computation might require thousands of transitions involving moving the gears and reading and writing symbols on the infinite tape.

Compared with modern computers, the Turing machine is a turtle. But surprisingly, whatever the hare can do (that is, whatever a fast computer can do), the turtle can do as well. Even the most advanced supercomputer programmed with the most sophisticated programming language can be programmed on Turing's simple machine with its roll, its head, and its simple gearbox. The wonderful thing about the Turing machine is that its operating manual is simple. I can describe it in half a page in a textbook, which makes it appealing to anyone who wants to prove anything theoretical about the limits or the power of computation. Thus, rather than having to understand how a complicated computer works or how a complicated programming language works,

they just have to think about this simple machine, because what this machine can do and what this machine cannot do match what can and cannot be done by the most powerful machine you might have.

What I have described is a mathematical construct. The Turing machine doesn't have any diodes or transistors or physical parts; it has a gear, but only on paper. In the play Turing is asked, "Are you going to build this machine?" "No," he replies, "I'm not going to build it." It is destined to live on paper, never to be built. But it is very much alive in the sense that it gives life to the computational process. That is, as the Turing machine reads a symbol from the tape and decides whether to shift gears and write something, as it moves step by step in this way it gives life to the computational process. If you observed the sequence of transitions, the shifting of gears and writing of symbols, you would see a frame-by-frame recording of the life of a particular computational process.

After defining the machine and the computational process, Turing asked what could be computed using this computational process. He defined computable tasks to be those problems for which one can make up a machine of this sort (that is, a machine designed to solve a computational task). The machine would go through a group of transitions and then output the solution; it would say, "Here. I've solved the task. It's done."

His thesis was that anything that is computable can be computed by this kind of a process. (Sometimes it is called the Church-Turing thesis because Alonzo Church's lambda calculus, which was invented independently, was shown to be equivalent to Turing machines.)

The Turing thesis is just that – a thesis. There is no proof that it is true or not true, because Turing took something informal – anything that is effectively computable,

which is just an informal intuition – and said it was equal to something formal that he defined. His claim is that anything that can be computed (that is, this informal stated thing) is equivalent to what his machine can compute.

However, as powerful as this machine seems to be – it is equivalent to any other computer that we have – it cannot do everything. This is one of the most fascinating aspects of Turing's work. A fundamental theorem of computer science is that certain well-defined tasks are not computable on a Turing machine and therefore not computable at all. Specifically, Turing showed that the task of computing whether a Turing machine will not be able to compute something is not computable.

What is unique and amazing about Turing's formulation of the Turing machine is not just that it captured computation or the process of computation. Church did so, too, with his lambda calculus, but he was build-

ing on the work of David Hilbert, Kurt Gödel, and other giants of mathematics. Church thus defined computation as a mathematician would. Turing, on the other hand, defined computability in terms of a machine, an engine that has tape and state transitions (that is, shifting “gears”). What he came up with was very different from what mathematicians had been doing and what Church did.

Turing’s solution is also simple, natural, intuitive. In fact, that is why today we refer to Turing computability rather than Turing-Church computability and why most undergraduate texts in computer science explain computability in terms of a Turing machine and what it can compute.

Turing’s 1937 definition of computation in terms of a machine is not just another mathematical formulation; it is the birthplace not only of the abstract notion of a computer but of the scientific discipline we call computer science.



Silvio Micali

Silvio Micali is Ford Professor of Engineering at the Massachusetts Institute of Technology. He was elected a Fellow of the American Academy in 2003.

Turing was a great man because he gave us something new and then gave us the means to go beyond what he had done. He allowed those of us who came later to have further growth. And that is the best thing we can aspire to as humanists, scientists, and, simply, human beings.

What was the point of rephrasing computation? Church and others had already defined it in terms of lambda calculus, recursive functions, and other things. Why define it in terms of machines? Because doing so ultimately allows us to speak about complexity. Some computations are easy. Some are harder, more complex. Some are harder still. And when some computations are too complex they are de facto impossible. Differentiating what can be computed efficiently from what cannot has allowed us to transform computer science and mathematics. Once you slap the lens of complexity on top of things, they look very different. Had computability been defined without machines, it would not have given birth to complexity.

At the beginning of *Breaking the Code*, Turing asks, “What is mathematics? It’s about

right and wrong.” Well, it used to be. Things have changed a little because the line between right and wrong has been eroded – ironically, thanks to Turing.

Why do we prove theorems? Because we want somebody to read our proofs in order to verify our work. Why do we read proofs? Because reading proofs, verifying the work of others, is simpler than discovering the proof in the first place. The gap between the harder task of discovering the proof and the easier task of reading and verifying the proof is implicit, and is necessary formally in order to prove that it is a separately meaningful concept, and not just another term for discovering truth. Verifying a proof ought to be simpler than finding it, else we can just publish theorem statements. Want to verify that they are correct? Prove them yourself!

Turing’s wonderful notion of a machine allows us to define what can be called a proof. At the beginning of the play, Turing explains what a proof is. He describes it in classical terms. But he gave us the means to go beyond the classical. The classical term is still intact: a proof is a string of symbols; you inspect each one in order to figure out whether a certain statement is true or false. But is this the best way to explain, to develop understanding? Consider what happens in school classrooms. Students ask questions and the teacher answers. The teacher doesn’t simply say, “Read this book,” and then the next year, “Read this other book,” and so on. If children could learn simply by reading books, our need for so many teachers would plummet. We would save enormous amounts on the cost of education. But education requires interaction, and interaction, the process of education, is intensive.

The simple way of educating students – handing them one book after another – is not nearly as efficient as the interactive method, which requires dialogue. This tells us that if indeed complexity is important then we should investigate other ways of proving things than just writing down their classical proofs. Proofs need no longer be confined to be syntactic objects; they can be

interactive processes. In this way we might efficiently grab more truth than before.

What about this business of right and wrong? At what point should we be satisfied with a proof? Assume somebody gives you a proof but he may be lying about whether it works. If you can catch him only about half the time, that's not so good, because half the time you will be fooled. But what if the chance he is lying is only one in four? That is slightly better. What if the chance he is lying is one in two to a thousand (1 in 2^{1000})? The chance he is lying is still real, but for all practical purposes it is essentially zero. The probability that he is not lying is over-

Another offshoot of Turing's work, one that has well served computer scientists and, especially, cryptographers, is the notion of computational indistinguishability. One of the things Turing is famous for is the so-called Turing test in which one person engages in a conversation with a machine and a second person and tries to determine which is which. The test represents a great insight because it suggests that rather than an ontological description of something we can use a procedural description.

Imagine you go to a jewelry store and find a sizable diamond for sale. You say, "This is a beautiful diamond. How much is it?" The

"That's right."

"Okay. I don't care. Give me two of the \$10 diamonds."

That's the Turing test. What advantage do you have in equating things that are not really the same? Well, you can then do things like equate encryption of zero and encryption of one, giving you an essentially unbreakable code, because they look the same no matter what you do in efficient time. You can have a notion of pseudo-randomness, computers that can expand, say, a thousand random bits to one trillion bits that aren't truly random, but nobody can tell them apart. No statistical test whose results you

can see in your lifetime or in the lifetime of the universe will give you different results. If that is the case, do you care that the bits are not truly random?

What Turing did is to achieve great clarity. Then he gave us complexity. Once you start ap-

plying the notion of complexity in machines to everything else that you know, the world begins to look simultaneously the same and different. Much of the mathematics and computer science we are working on today and will be working on in the future is going to pass through the lens this extraordinary man has given us. This notion of extreme simplicity has opened a world in which different things look equal and the world is actually better because of it. ■

© 2011 by Alan Lightman, Laurence Sene-lick, Ronald L. Rivest, Shafi Goldwasser, and Silvio Micali, respectively



To view or listen to the presentation, visit <http://www.amacad.org/events/breakingCode/code.aspx>.

The Turing test represents a great insight because it suggests that rather than an ontological description of something we can use a procedural description.

whelming. Perhaps we should be happy with that. If we now agree to accept as true things that might be false only with an overwhelmingly small probability, the range of problems that are efficiently provable becomes gigantic.

And why stop at that? We can now also have a theory of mathematics that is deliberately inconsistent. *Breaking the Code* starts with Turing saying, "I want consistency. And I want completeness. I want every theorem to be provable, but I never want to prove A and the opposite of A. Otherwise, what's a proof good for?"

Complexity gives us a strange way out. Imagine a system of proof in which we can prove A and we can also prove not-A. But there is a caveat: if A is true, in the classical sense, then a "good-looking" proof of not-A either does not exist, which is the best of all possible worlds, or it exists but takes billions and billions and billions and billions of years to discover. Which one would you accept? You would accept the first option. The consistency Turing so desperately wanted turns out not to be so crucial.

jeweler answers, "\$100,000." But then you see another diamond that seems to be the equal of the first; it looks just like the \$100,000 diamond. The jeweler says, "Ah, but the two are not equal."

"Oh, how much is the second diamond?" you ask.

"\$10."

"But it looks the same."

"Yes, of course it looks the same."

"But it weighs the same."

"Of course it weighs the same."

"What if I put it under a microscope?"

"It still looks the same."

"What if I put it in the microwave for ten hours?"

"Still the same."

"What if I examine it with an MRI?"

"Still the same."

"What if -"

"Sir, no matter what you do in your lifetime, or the lifetimes of your descendants until the sun runs cold - no matter what experiments you do - the two diamonds will look the same."

"But they are not the same?"