CARLOSCASTILLA / GETTY IMAGES / ISTOCK

KRYPTOGRAFIE
DER HEILIGE GRAL DER INFORMATIK

Kann man ein Computerprogramm so verschleiern, dass sich dessen Funktionsweise nicht vorhersagen lässt? Forscherteams der Mathematik und Informatik haben sich dieser anspruchsvollen Aufgabe gestellt – und dabei eine Methode gefunden, an die keiner mehr geglaubt hatte.

Christian Matt forscht beim Blockchain Start-Up Concordium in Zürich.

spektrum.de/artikel/1937242

CARLOSCASTILLA / GETTY IMAGES / ISTOCK

KAUDERWELSCH Obfuskatoren verwandeln Programmcodes in unverständliche Zeichenfolgen – doch ein allgemein gültiges Verfahren war lange nicht in Sicht.

AUF EINEN BLICK
ALGORITHMISCHE VERSCHLEIERUNG

1 Damit Programme nicht kopiert oder verändert werden, versuchen Informatiker, ihre Funktionsweise zu verschleiern, ohne ihre Funktionalität zu beeinträchtigen.

2 Ein kryptografisches Verfahren hierfür zu entwickeln, das für alle Algorithmen funktioniert, ist aber sehr schwer. Tatsächlich waren sämtliche bisherigen Ansätze gescheitert.

3 Deshalb gaben viele Fachleute die Hoffnung auf. Nun haben einige aber bewiesen, dass es eine allgemein gültige Methode gibt, die Programme nachweislich sicher verschlüsselt.

Softwarehersteller sind seit jeher daran interessiert, Programme zu entwickeln, die ein Computer zwar ausführen kann, deren genaue Funktionsweise aber nicht direkt ersichtlich ist. Das hat verschiedene Gründe: Einerseits verhindert man, dass die Konkurrenz die Software kopiert und in ihr eigenes Produkt einbaut. Andererseits können Kriminelle sie nicht manipulieren und so beispielsweise einen Kopierschutz umgehen.

Daher haben Programmiererinnen und Programmierer schon zahlreiche Methoden entwickelt, um Algorithmen zu verschleiern, ohne ihre Funktionalität zu verändern. Das Vorgehen ist als Obfuskation bekannt. Auch in der Kryptografie erweist sich die Technik als überaus nützlich, da man auf diese Weise Geheimnisse in öffentlicher Software verstecken kann. Damit lassen sich zahlreiche kryptografische Probleme elegant lösen.

Ein sicheres Verfahren, das für alle Arten von Programmen zuverlässig funktioniert, war lange außer Reichweite. Viele bezweifelten sogar, dass ein solches überhaupt existiert. Doch das hat sich nun geändert: In einer 2020 erschienenen Forschungsarbeit ist es den Informatikern Aayush Jain und Amit Sahai von der University of California in Los Angeles zusammen mit ihrer Kollegin Huijia Lin von der University of Washington gelungen, eine allgemein gültige Verschleierung zu finden, die nachweislich sicher ist.

Um Computerprogramme zu entwickeln, verfasst man einen Quellcode in einer Sprache wie Java oder C++, die so entwickelt wurden, dass sie für Menschen gut verständlich sind. Andernfalls wäre die Arbeit äußerst mühsam, insbesondere wenn größere Teams gemeinsam daran beteiligt sind. Der Code wird dann von einem Compiler in eine für Computer lesbare Maschinensprache übersetzt. Diese hat das primäre Ziel, effizient ausgeführt zu werden, und ist in der Regel deutlich schwieriger zu entziffern als der Quellcode. Daher ist eine solche Übersetzung bereits eine Art von Obfuskation – wenn auch keine besonders sichere.

Denn inzwischen gibt es Dekompilierer, die Maschinencode wieder in eine lesbare Programmiersprache zurückübersetzen. Zwar können sie selten den ursprünglichen Quellcode wiederherstellen, doch meist ist das Ergebnis gut genug, um die Funktionsweise eines Algorithmus mit entsprechendem Aufwand nachzuvollziehen.

Deshalb gibt es eine Reihe kommerzieller Obfuskatoren, die das Dekompilieren erschweren. Gängige Methoden sind das Einfügen irrelevanter Berechnungen, die das Programm komplizierter machen, ohne das Ergebnis zu beeinflussen. Manchmal strukturiert man auch die Codezeilen um, wodurch viele überflüssige Sprünge ausgeführt werden. Eine weitere Möglichkeit ist, gezielt spezielle Befehle einzufügen, die Dekompilierern Probleme bereiten.

All diese Methoden führen jedoch bestenfalls zu einem Katz-und-Maus-Spiel, bei dem laufend neue Tricks verwendet werden, die einfallsreiche Hacker aber bald zu umgehen lernen und so weiter. Echte Sicherheit kann man auf diese Weise nicht gewährleisten – man erhöht nur den Aufwand, die Programme zu verstehen.

Daher haben sich Entwickler der Kryptografie zugewandt, einer mathematischen Disziplin innerhalb der Informatik, die sich mit sicherer Kommunikation befasst. Seit der Antike versuchen Gelehrte, Texte zu verschlüsseln, damit Unbefugte sie nicht lesen. Die moderne Ausprägung umfasst jedoch viel mehr Gebiete als die Chiffrierung von Schriftstücken, sie bildet inzwischen unter anderem die Grundlage für Onlineshopping und -banking.

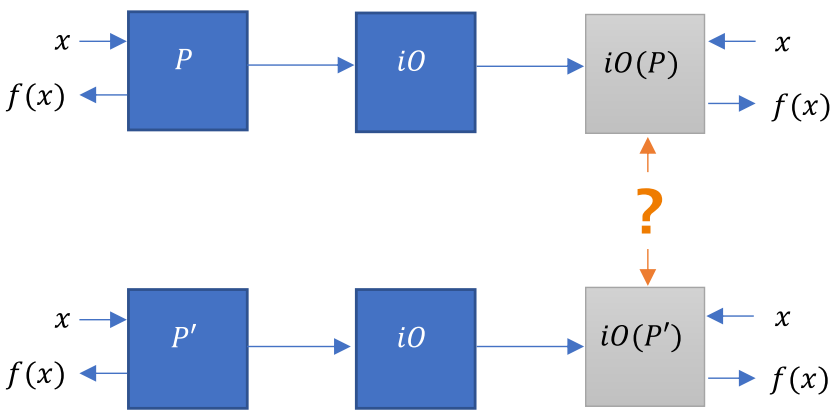
Kryptografische Obfuskation gibt sich nicht damit zufrieden, es einer Partei möglichst schwer zu machen, die Funktionsweise eines Algorithmus nachzuvollziehen. Ein auf diese Art verschleiertes Programm soll aus mathematischer Sicht – zumindest unter bestimmten Annahmen – unmöglich zu knacken sein. Mit einer solchen Verschleierung könnte man nicht nur Software sichern, sondern sogar neuartige Verschlüsselungen entwickeln.

Geheime Nachrichten austauschen

Um den letzten Punkt zu verstehen, muss man mehr über Chiffrierungsmethoden erfahren. Diese bestehen häufig aus zwei Algorithmen, einen zum Ver- und einen zum Entschlüsseln. Ersterer verwandelt eine Nachricht in ein Chiffrat, das der Sender einem Empfänger übermittelt. Dieser kann dann mit Hilfe eines Schlüssels den Inhalt entziffern.

Die einfachste Art der Chiffrierung stellen symmetrische Verfahren dar, die beim Dekodieren das gleiche Werkzeug nutzen wie zum Verschlüsseln. Über Jahrhunderte haben Militär und Geheimdienste auf diese Weise operiert, um vertrauliche Informationen zu übermitteln. Ein entscheidender Nachteil ist, dass Sender und Empfänger den Schlüssel vorab austauschen müssen. Ein solcher Ansatz ist gerade im Internet, wo Parteien sich nie treffen, nicht praktikabel.

Als Alternative schlugen 1976 die Informatiker Whitfield Diffie und Martin Hellman, damals beide an der Stanford University, das so genannte Public-Key-Verfahren vor, für das sie 2015 den Turingpreis bekamen, die angesehenste



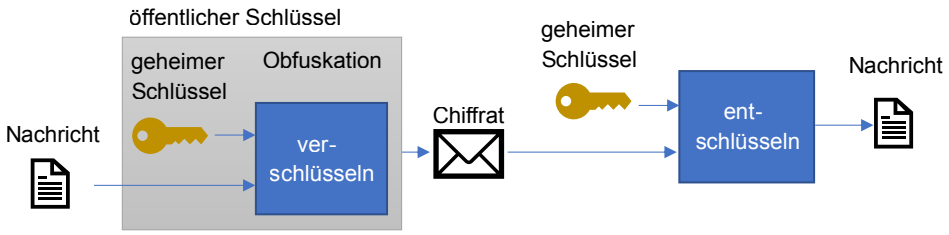
CHRISTIAN MATT

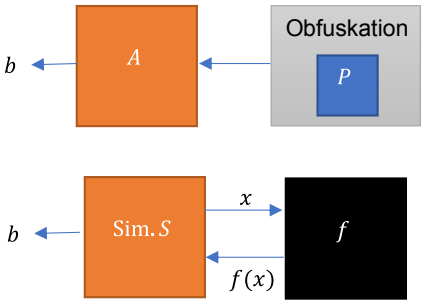
INDISTINGUISHABILITY OBFUSCATION **Wenn zwei Programme *P* und *P′* die gleiche Funktion *f* berechnen, verschleiert iO die zwei Programme derart, dass sie ununterscheidbar sind und weiterhin *f* berechnen.**

iO ist optimal

Angenommen, es gäbe einen Obfuskator O*, der ein Programm *P* besser verschleiert als iO. Demnach könnte man von iO(*P*) etwas erfahren, das durch O*(*P*) verborgen bleibt. Indem man einen neuen Algorithmus *P′*=O*(*P*) definiert, berechnen *P* und *P′* die gleiche Funktion, da Obfuskation die Funktionalität nicht verändert. Wendet man nun iO auf *P* und *P′* an, lassen sich die chiffrierten Programme nicht mehr voneinander unterscheiden.

Mit Hilfe von Obfuskation lassen sich symmetrische Verschlüsselungen in ein Public-Key-Verfahren umwandeln.





Virtual-Black-Box Obfuscation: Links extrahiert ein Angreifer *A* ein geheimes Bit *b* aus dem obfuskierten Programm. Der Obfuskator O erfüllt die starke Sicherheitsdefinition, wenn ein Simulator *S* die gleiche Information *b* nur mit Zugriff auf eine Black-Box zur Funktionsauswertung ermitteln kann.

Daraus folgt aber, dass man aus iO(*P′*) die gleichen Informationen extrahieren kann wie aus iO(*P*). Wäre das nicht der Fall, ließen sich iO(*P*) und iO(*P′*) auf diese Weise differenzieren. Das würde bedeuten, dass *P′*=O*(*P*) ebenfalls die entsprechende Eigenschaft preisgibt, was der ursprünglichen Annahme widerspricht. Daher existiert kein Obfuskator, der sicherer ist als iO.

CHRISTIAN MATT

Auszeichnung ihres Fachs. Die asymmetrische Technik basiert auf zwei verschiedenen Schlüsseln: einen zum Codieren und einen zum Decodieren. Das erlaubt es, Ersteren in einem zugänglichen Verzeichnis oder auf einer Webseite zu speichern. Nur der Decodierungsschlüssel bleibt verborgen.

Jede Person besitzt demnach zwei Schlüssel, einen öffentlichen und einen geheimen. Um eine Botschaft zu versenden, sieht man den öffentlichen Schlüssel ein, chiffriert die Nachricht damit und übermittelt sie. Obwohl beide vorher keine Information ausgetauscht haben, kann nur der Empfänger den codierten Inhalt mit Hilfe seines verborgenen Schlüssels entziffern. Dieser revolutionäre Ansatz ist im Internet inzwischen allgegenwärtig.

Diffie und Hellman erkannten damals zwar das Potenzial des Verfahrens, konnten jedoch keine konkrete Umsetzung vorstellen. Sie beschrieben aber die Möglichkeit, eine symmetrische Verschlüsselung mittels Obfuskation in eine Public-Key-Technik zu verwandeln: Dazu verschleiert man die Funktion, die eine Nachricht chiffriert, und publiziert das Ergebnis als öffentlichen Schlüssel. Allerdings war damals keine derartige Verschleierung bekannt, wodurch auch diese Methode vorerst nur ein Gedankenexperiment blieb.

Die Umsetzung eines Public-Key-Verfahrens ließ nicht lange auf sich warten: Bereits zwei Jahre nach Diffies und Hellmans Veröffentlichung entwickelten die damals am Massachusetts Institute of Technology (MIT) angestellten Informatiker Ron Rivest, Adi Shamir und Leonard Adleman die so genannte RSA-Verschlüsselung. Diese wird zum Beispiel in heutigen Internetbrowsern verwendet.

Zwar hatte man nun eine sichere asymmetrische Verschlüsselungsmethode, dennoch ließ Fachleute der Gedanke an Obfuskation nicht los. Sie suchten weiterhin nach einer geeigneten Methode, Algorithmen zu verschleiern.

Ein erster Schritt besteht darin, das Problem einzugrenzen und von irrelevanten Details zu abstrahieren. Es ist beispielsweise hilfreich, sich zunächst auf möglichst einfache Programme zu beschränken, die aber immer noch komplex genug sind, um die gewünschten Anwendungen zu gewährleisten, etwa die Konstruktion von Public-Key-Verschlüsselungen. Beispiele dafür sind Algorithmen, die eine einzige Funktion berechnen. Sie erhalten einen Wert *x* und geben *y* = *f*(*x*) aus. Interaktionen mit dem Nutzer, Ausgaben auf dem Bildschirm und Ähnliches blendet man hier bewusst aus. Anschließend kann man sich dem Obfuskator zuwenden: einem Algorithmus O, der ein Programm *P* in ein anderes (O(*P*)) umwandelt, wobei *P* und O(*P*) die gleiche Funktion berechnen.

Bisher hat man jedoch noch nichts über die Sicherheit des Verfahrens ausgesagt. Zum Beispiel erfüllt ein Obfuskator, der alles beibehält (für den also O(*P*) = *P* gilt), die vorangehende Bedingung. Damit ist aber nichts verschleiert.

Grundpfeiler von iO

Multilineare Abbildungen sind Funktionen, die viele Objekte einer einzigen Zahl zuordnen: $f(x_1, x_2, ..., x_n) = z$. Zudem erfüllen sie folgende Eigenschaft:
$f(x_1,..., x_i \cdot y_i,..., x_n) = f(x_1,..., x_i,..., x_n) \cdot f(x_1,..., y_i,..., x_n)$.
Mit ihnen lassen sich bestimmte Codierungen definieren. Wenn man ein Objekt a hat und dieses verschlüsseln möchte, lassen sich für eine multilineare Abbildung n-ten Grades n+1 Codierungen bestimmen: $[a]_1$, $[a]_2$,...,$[a]_{n+1}$.

Die Annahme, die Kryptografen für dieses Verfahren treffen, ist folgende: Man kann die Zahl a nicht effizient aus $[a]_i$ zurückrechnen. Zudem lässt sich die Chiffrierung $[a \cdot b]_i$ nicht von der einer zufälligen Zahl $[r]_i$ unterscheiden, wenn man nur die Werte $[a]_i$ und $[b]_i$ kennt. Letzteres ist die so genannte Decisional-Diffie-Hellman-Annahme und geht zurück auf die 1976 erschienene Arbeit der beiden Informatiker Whitfield Diffie und Martin Hellman zur Public-Key-Verschlüsselung.

Daher ist es entscheidend, sich darauf zu einigen, was »sicher« in diesem Kontext überhaupt bedeutet. Gute Definitionen zu finden, erweist sich häufig als erstaunlich kompliziert. 2001 widmete sich der Informatiker Satoshi Hada von IBM Research sowie ein Jahr später ein anderes Forscherteam um Boaz Barak von der Princeton University der anspruchsvollen Aufgabe.

Da das verschleierte Programm O(*P*) die Funktionalität von *P* erhält, ist klar, dass O(*P*) nichts verstecken kann, was man durch wiederholtes Ausführen von *P* lernen kann. Bestimmt *P* beispielsweise die Quadratwurzel einer Zahl, kann O(*P*) nicht verschleiern, dass bei der Eingabe 9 der Wert 3 herauskommt. Man kann aber hoffen, dass O(*P*) nur diese Tatsache preisgibt und nicht, auf welche Weise es das Ergebnis berechnet.

Eine undurchsichtige Box

Eine solche Sicherheitsgarantie haben Barak und seine Kollegen durch die so genannte Virtual-Black-Box-Obfuskation formalisiert. Dazu betrachtet man einen Angreifer A (ein effizienter Algorithmus), der etwas über das verschleierte Programm O(*P*) erfahren möchte, das eine Funktion *f* auswertet. O gilt als sicher, falls ein Simulator S (ebenfalls ein effizienter Algorithmus) existiert, der die gleiche Information nur durch wiederholtes Ausführen von *f* ermitteln kann. Das heißt, O(*P*) ähnelt einer Black Box, welche die Funktion *f* auswertet.

Mit dieser Vorarbeit begann die Suche nach sicheren Obfuskatoren, doch die Bemühungen endeten schnell. Denn das Forscherteam um Barak bewies, dass eine solche Technik nicht existiert. Dazu konstruierten sie ein spezielles Programm *P*, für das jede mögliche Obfuskation O(*P*) mehr preisgibt als der Zugriff auf eine Black Box, welche die dazugehörige Funktion auswertet. Auch wenn P explizit zu diesem Zweck erstellt wurde und keinerlei praktischen Bezug hat, zeigt das Ergebnis dennoch, dass die Suche nach einer allgemeingültigen Virtual-Black-Box-Obfuskation aussichtslos ist.

Offenbar war der Sicherheitsbegriff zu stark, um ein geeignetes Verfahren dafür zu entwickeln. Deshalb schlugen die Forscher um Barak eine schwächere Variante vor, die so genannte ununterscheidbare Obfuskation (kurz: iO von indistinguishability Obfuscation): Wenn zwei Programme *P* und *P′* die gleiche Funktion *f* berechnen, dann lassen sich die verschleierten Versionen iO(*P*) und iO(*P′*) nicht voneinander unterscheiden. Das heißt, es gibt keinen effizienten Algorithmus, der herausfinden kann, ob er es mit iO(*P*) oder iO(*P′*) zu tun hat.

Im Gegensatz zur Virtual-Black-Box-Obfuskation erscheint iO auf den ersten Blick nicht besonders nützlich. Schließlich möchte man in der Regel nur ein Programm verschleiern und nicht zwei verschiedene, welche die gleiche Funktion berechnen. Außerdem sollte eine sichere Obfuskation möglichst wenig – bis gar nichts – über die Funktionsweise von *P* preisgeben. iO(*P*) und iO(*P′*) könnten aber beide viel offenbaren, ohne dass sie der Definition von iO widersprechen.

Wenn man jedoch das Public-Key-Verfahren betrachtet, wird der Vorteil von iO klar. Startet man mit einer symmetrischen Verschlüsselung, kann man den entsprechenden Schlüssel in einem Programm *P* verstecken und dieses anschließend obfuskieren (iO(*P*)). Die ununterscheidbare Obfuskation garantiert, dass der Schlüssel darin genau so gut versteckt ist wie in allen anderen Algorithmen *P′*. Damit ist iO(*P*) so sicher wie das beste Verschlüsselungsprogramm *P′*.

Das ist zwar eine gute Nachricht, doch wenn es um die praktische Umsetzung geht, ist man nicht wirklich weiter gekommen. Um die Sicherheit von iO(*P*) zu garantieren, müsste man beweisen, dass es ein Programm *P′* gibt, das den Schlüssel perfekt versteckt. In diesem Fall könnte man aber *P′* direkt verwenden und ganz auf Obfuskation verzichten. Erstaunlicherweise kann man zeigen, dass iO die beste allgemeingültige Möglichkeit ist, um Algorithmen zu verschleiern (siehe »iO ist optimal«). Allerdings sagt auch das nichts über den tatsächlichen Nutzen von iO aus.

Barak und seine Kollegen konnten die ununterscheidbare Obfuskation in ihrer Arbeit zwar definieren, fanden aber kein konkretes Beispiel dafür. Weil die Anwendungsgebiete zudem fraglich waren, verloren viele Kryptografinnen und Kryptografen das Interesse an dem Thema. Das änderte sich jedoch 2013 schlagartig, als Forscher um den Informatiker Sanjam Garg von IBM Research nicht nur einen Kandidaten für iO fanden, sondern auch eine Möglichkeit präsentierten, mit dessen Hilfe ein bis dahin ungelöstes Problem der Kryptografie anzugehen.

Dieses hängt mit so genannter Functional Encryption zusammen, einer Technik, die nur einen Teil der codierten Informationen preisgibt, während der Rest geheim bleibt. Das Verfahren ermöglicht es, Schlüssel zu konstruieren, die verschiedene Eigenschaften der ursprünglichen Nachricht *x*

offenbaren. Damit unterscheidet sich der Ansatz von gewöhnlicher Verschlüsselung: Dort erfährt man entweder alles (wenn man den geheimen Schlüssel hat) oder nichts (wenn man ihn nicht besitzt).

Die Methode ermöglicht es, den Zugriff auf chiffrierte Daten präzise zu steuern. Zum Beispiel lässt sich für verschlüsselte E-Mails eine Funktion generieren, die angibt, ob es sich bei der Nachricht mit hoher Wahrscheinlichkeit um Spam handelt. Die Funktion kann man dem E-Mail-Anbieter zur Spamfilterung geben, ohne die Vertraulichkeit des Inhalts zu beeinträchtigen.

Ein plötzlicher Hype um Obfuskatoren – doch ohne konkrete Umsetzung

Vor der Arbeit von Garg und seinen Kollegen waren nur wenige Funktionsklassen bekannt, für die man das Functional-Encryption-Verfahren einsetzen konnte. Wie die Wissenschaftler aber herausfanden, könnte man mit iO erstmals eine Methode konstruieren, die beliebige Funktionen unterstützt – und damit verschiedenste Informationen einer Nachricht extrahiert (siehe »Functional Encryption dank iO«).

Das Interesse an ununterscheidbarer Obfuskation stieg daraufhin wieder an. Einerseits erschienen viele Arbeiten, die weitere iO-Kandidaten vorschlugen, andererseits zeigten Fachleute, dass man damit immer mehr Probleme lösen könnte. Zum Beispiel bewiesen Sahai und sein Kollege Brent Waters von der University of Texas 2014, wie man mittels iO eine symmetrische Verschlüsselung in ein Public-Key Verfahren umwandeln kann – was vorher nur für eine Virtual-Black-Box Obfuskation bekannt war.

Dabei konzipierten Sahai und Waters spezielle Tricks, die es ermöglichten, weitere Probleme zu bearbeiten. Schnell wurde klar, dass sich mit Hilfe von iO viele offene Fragen in der Kryptografie beantworten lassen.

Während es immer mehr Anwendungen und Kandidaten für iO gab, war nichts davon allgemein beweisbar sicher. Es ist zwar nicht schwer, für konkrete Beispiele entsprechende Obfuskationen zu konzipieren, aber eine Methode zu finden, die für alle möglichen Programme funktioniert, erweist sich als überaus kompliziert. Die Suche danach wurde zu einer der wichtigsten Aufgaben des Bereichs.

In der Kryptografie gilt ein Verfahren als sicher, wenn man zeigt, dass es keinen Algorithmus gibt, der es knacken kann. Auf iO bezogen würde es bedeuten, dass kein Programm iO(P) von iO(P') unterscheiden kann. Das zu beweisen ist extrem komplex: Man muss für alle möglichen Algorithmen nachweisen, dass sie an der Aufgabe scheitern. Vor diesem Problem stehen Kryptografen nicht nur bei Obfuskatoren, sondern auch schon bei vergleichsweise simplen Verschlüsselungsverfahren.

Um weiterzukommen, stützt man sich daher auf Annahmen. Zum Beispiel jene, dass es schwierig ist, große Zahlen in ihre Primfaktoren zu zerlegen. Demnach gibt es keinen effizienten Algorithmus, der für eine natürliche Zahl N eine Liste von Primzahlen ausgibt, deren Produkt N ergibt. Jedoch kann niemand mit Sicherheit sagen, ob ein solches Programm nicht doch existiert – und bisher nur noch nicht gefunden wurde. Weil sich zahlreiche Mathematiker und Physiker schon seit Jahrzehnten dem Problem erfolglos verschrieben haben, erscheint die Annahme aber plausibel.

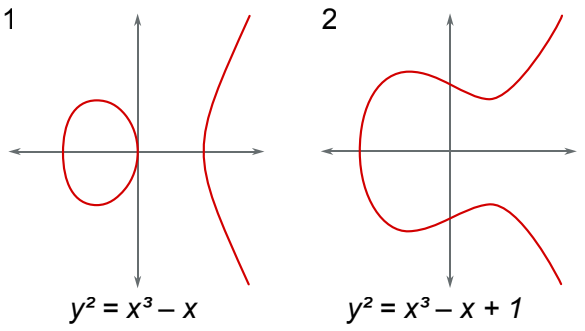
Um die Sicherheit einer Methode zu beweisen, verwendet man typischerweise Widerspruchsbeweise: Man nimmt an, es gäbe einen effizienten Algorithmus A, der das Verfahren brechen kann. Aus einem derartigen A versucht man dann, ein weiteres Programm A' zu konstruieren, das zum

Elliptische Kurven

Diese Art von Funktionen hat in den letzten Jahrzehnten eine bedeutende Rolle in der Mathematik gespielt. Die wohl größte Errungenschaft, zu der sie beigetragen hat, ist der Beweis des großen fermatschen Satzes.

Allgemein lassen sich elliptische Kurven in folgender Form schreiben: $y^2 = x^3 + ax + b$. Das Besondere an ihnen ist unter anderem ihre symmetrische Struktur. Addiert man zwei Punkte auf der Kurve, landet man zwar – anders als bei Geraden – außerhalb der Kurve. Indem man aber eine abgewandelte Form der Addition definiert, lassen sich Punkte so miteinander verknüpfen, dass das Ergebnis wieder auf der Kurve liegt.

Die Eigenschaft kann man auch in der Kryptografie nutzen: Bei asymmetrischen Verschlüsselungsverfahren sucht man stets nach Operationen, die sich einfach berechnen, aber nur schwer umkehren lassen – so wie die Primfaktorzerlegung: Man kann schnell überprüfen, ob das Produkt zweier Primzahlen mit



einem Wert übereinstimmt, wohingegen es überaus aufwändig ist, große Zahlen in ihre Primfaktoren zu zerlegen. Ebenso verhält es sich mit Punkten auf elliptischen Kurven: Für eine Zahl n und einen Punkt P lässt sich $n \cdot P = A$ sofort bestimmen, doch wenn A und P bekannt sind, kann man nur sehr schwer den Wert n ermitteln. Daher eignen sich elliptische Kurven für Public-Key-Verfahren.

Beispiel Zahlen schnell in ihre Primfaktoren zerlegt. Da solche aber nicht existieren (zumindest soweit man weiß), kann es auch A nicht geben. Die Methode zu knacken, ist daher mindestens so schwierig wie das Faktorisieren großer Zahlen.

Die ersten Kandidaten für iO funktionierten wie ein extrem komplexes mehrdimensionales Puzzle: Man zerlegt ein Programm in viele Bestandteile und verschleiert jeden davon, indem man zufällige Elemente hinein mischt. Deren Sicherheit basierte jedoch nicht auf bewährten Annahmen, sondern auf neu formulierten, die alle widerlegt wurden. Das stimmte viele Fachleute skeptisch, ob iO überhaupt existierte – eventuell war sie genau wie Virtual-Black-Box-Obfuskation unmöglich zu realisieren.

Die neuartigen Annahmen der frühen Kandidaten hingen mit so genannten multilinearen Abbildungen (siehe »Grundpfeiler von iO«) zusammen. Dabei handelt es sich um Funktionen vom Grad d, die d Eingabewerte einer Zahl zuordnen. Fachleute setzten voraus, dass man mit deren Hilfe Objekte verschlüsseln kann – das ist die Annahme, auf der letztlich die Sicherheit des Verfahrens beruht: Man zerlegt das Programm und chiffriert die einzelnen Bestandteile durch solche Abbildungen. Doch wie sich herausstellte, lassen sich viele multilineare Abbildungen knacken. Bisher kennt man nur für bilineare Abbildungen, die zwei Werten eine Ausgabe zuweisen, eine sichere Realisierung – und zwar mit Hilfe elliptischer Kurven (siehe »Elliptische Kurven«).

Daher suchten Forscherinnen und Forscher nach Möglichkeiten, iO mit multilinearen Abbildungen niedrigen Grads zu definieren, das heißt für solche, die nur wenige Eingabewerte erhalten. Denn je geringer der Grad, desto höher schien die Chance auf eine Umsetzung. Die Informatikerin Huijia Lin und ihr Kollege Stefano Tessaro von der University of California in Santa Barbara reduzierten 2017 den benötigten Grad auf drei. Man stand kurz vor dem Ziel, doch der letzte Schritt schien unerreichbar.

Verschiedene Sicherheitskonzepte

Die sicheren iOS basieren aber nicht nur auf multilinearen Abbildungen, sondern auch auf Zufallszahlen, die mit so genannten Pseudozufallsgeneratoren, kurz PRG (aus dem Englischen: pseudorandom generator) berechnet werden. Dabei handelt es sich um Funktionen, die eine Eingabe (Seed) erhalten und diese zu einem längeren Ausdruck expandieren. Sofern man den Seed nicht kennt, soll sich die Ausgabe nicht von völlig zufälligen Werten unterscheiden. Weil Abbildungen stets deterministisch sind, können sie jedoch keinen echten Zufall erzeugen, daher nennt man sie pseudozufällig.

Ein einfaches Beispiel ist ein PRG, der einen aus zwei Werten bestehenden Seed zu drei Zahlen expandiert, wobei die dritte der Summe der Eingabewerte entspricht: $\mathrm{PRG}(s_1, s_2) = (s_1, s_2, s_1 + s_2)$. Für sich betrachtet sehen die drei Ausgaben zufällig aus. Zusammengenommen erkennt man aber schnell das Schema dahinter. Ein sicherer PRG

Functional Encryption dank iO

Die Grundidee der Functional Encryption besteht darin, zunächst eine Nachricht x mit einem gewöhnlichen Public-Key-Verfahren zu codieren: Man generiert einen geheimen Schlüssel S und einen öffentlichen P. Anschließend erzeugt man mit Hilfe von S einen zusätzlichen verborgenen Schlüssel S_f für eine Funktion f. Dann verschlüsselt man den Inhalt x mit dem publiken Schlüssel $P(x)$ und erhält eine chiffrierte Nachricht c. Mit S_f kann man aus c die Funktion $f(x)$ ermitteln: $S_f(c) = f(x)$. Die Schwierigkeit besteht darin, geeignete S_f zu finden.

Gäbe es einen Virtual-Black-Box-Obfuscator, könnte man diesen nutzen, um S_f zu konstruieren. Dafür geht man am besten rückwärts vor: Man möchte aus c die Funktion $f(x)$ extrahieren. Das heißt, man muss den chiffrierten Text c entschlüsseln, um x zu erhalten. Das geschieht mit Hilfe des geheimen Schlüssels S: $S(c) = x$.

Um daraus $f(x)$ zu berechnen, wendet man also f auf beide Seiten der Gleichung an: $f(S(c)) = f(x)$. Im Prinzip hätte man damit schon einen Schlüssel, nämlich das Hintereinanderausführen von S und f. Doch dieser wäre nicht sicher – aus $f(S)$ ließe sich schnell ermitteln, wie S aussieht, und gerade das soll ja geheim sein. Daher verschleiert man die verschachtelte Funktion $f(S)$, indem man Obfuskation nutzt. Somit ist gewährleistet, dass $O(f(S(c)))$ die Ausgabe $f(x)$ liefert, aber es ist unmöglich, daraus Rückschlüsse auf S zu erhalten.

Doch leider ist Black-Box-Obfuskation nicht erreichbar. Daher nutzt man einen Trick, um mit iO ebenfalls einen sicheren Schlüssel für eine Funktion f zu entwickeln. Dazu chiffriert man ein Objekt x mit zwei verschiedenen Public Keys, so dass man c_1 und c_2 erhält. Mit einem so genannten Zero-Knowledge-Beweis π zeigt man dann, dass die Ergebnisse zur gleichen Nachricht gehören. Ein solcher Beweis verrät nichts über die ursprünglichen Inhalte, außer dass sie identisch sind.

Im ersten Schritt prüft man, ob π korrekt ist, und entschlüsselt in diesem Fall c_1 mit dem verborgenen Schlüssel S_1. Danach wendet man f auf die decodierte Nachricht x an und gibt das Resultat $f(x)$ aus. Da der geheime Schlüssel S_2 von c_2 nicht bekannt ist und zudem $iO(f(S_1))$ nicht von $iO(f(S_2))$ unterscheidbar ist, bleiben die Schlüssel sicher versteckt.

Learning Parity with Noise

Die bekannten Learning Parity with Noise (LPN)-Probleme handeln von Berechnungen in einem Zahlenraum, der einer Art Uhrzeit-Arithmetik folgt. Das heißt, alle Zahlen können bloß Werte zwischen 0 und $p - 1$ annehmen. Sollte man ein Ergebnis erhalten, das $p - 1$ überschreitet (etwa $p + 2$), beginnt man wie bei der Uhrzeit wieder bei null und zählt entsprechend hoch (aus $p + 2$ wird 2, wie 11 Uhr plus 3 Stunden 2 Uhr ergibt).

LPN-Probleme betrachten Zahlenräume, die bis zu einer Primzahl p gehen. Damit definiert man eine Matrix A und einen Vektor s mit zufälligen Einträgen in $\{0 \ldots p - 1\}$, zudem einen weiteren Vektor e, der hauptsächlich den Wert 0 hat. Anschließend berechnet man $b = As + e$.

Die Aufgabe besteht nun darin, für eine vorgegebene Matrix A den Vektor b von einem mit zufälligen Einträgen aus $\{0 \ldots p - 1\}$ zu unterscheiden. Bis heute zählt das unter Fachleuten zu einem schwierigen Problem, denn b lässt sich – wenn überhaupt – nur unter extrem hohem Aufwand von einem Zufallsvektor differenzieren.

muss daher deutlich komplexer sein, um die Zusammenhänge zwischen den Werten zu verstecken. Der Grad d eines PRG entspricht dem der Polynome, die seine Ausgaben beschreiben. Im vorigen Beispiel haben sie jeweils den Grad eins. Je höher der Grad, desto wahrscheinlicher lässt sich ein PRG realisieren, da die Ergebnisse komplexer ausfallen können und so schwerer von echten Zufallswerten zu unterscheiden sind.

Um eine sichere iO zu konstruieren, brauchte man ein PRG von Grad zwei, gepaart mit einer bilinearen Abbildung. Doch es gab ein Problem: Während für Polynome von Grad drei plausible PRGs existieren, deuten verschiedene Ergebnisse darauf hin, dass es keine für Grad zwei gibt.

Damit schien auch dieser Ansatz zum Scheitern verurteilt. Doch 2019 schlug ich zusammen mit Prabhanjan Ananth vom Massachusetts Institute of Technology sowie Lin, Jain und Sahai eine neue Art von Pseudozufallsgeneratoren vor. Sie liegen in zwischen solchen von Grad zwei und drei – man kann sie informell als vom Grad 2,5 ansehen.

Die entscheidende Idee war dabei, nur einen Teil des Seeds geheim zu halten und den Rest öffentlich zu machen. Der Ausgabewert lässt sich dann berechnen, indem man ein Polynom vom Grad zwei mit dem verborgenen Seed auswertet, während mit dem bekannten Stück komplexere Operationen erlaubt sind.

Ein Beispiel dafür ist $\mathrm{PRG}(s_1, s_2, p_1) = (s_1, s_2, p_1, s_1 \cdot s_2 \cdot p_1)$, wobei s_1 und s_2 zum geheimen und p_1 zum öffentlichen Teil des Seeds gehören. Insgesamt hat man den Grad drei, das verborgene Stück mit $s_1 \cdot s_2$ aber nur den Grad zwei. Den geheimen Teil kann man daher mittels bilinearer Abbildungen zu einem sicheren Verfahren verbinden und den publiken Teil anschließend hinzufügen.

Damit hatten wir eine Lösung gefunden, um eine ununterscheidbare Obfuskation zu realisieren. Doch es gelang uns in dieser Arbeit nicht, ein explizites Beispiel für einen PRG mit den benötigten Sicherheitseigenschaften zu konstruieren.

2021 fanden Jain, Sahai und Lin das fehlende Puzzlestück: einen PRG vom Grad 2,5. Dabei stützten sie sich auf bewährte Annahmen, wonach die so genannten »Learning Parity with Noise Problems« schwierig zu lösen seien (siehe »Learning Parity Problems«). Diese stammen aus der Codierungstheorie und werden schon seit Jahrzehnten untersucht, was Fachleute inzwischen davon überzeugt hat, dass es sich um schwere Probleme handelt.

Mit einem konkreten Beispiel für PRGs vom Grad 2,5 in der Hand konnten die Forscher dieses mit den bereits bekannten Realisierungen von bilinearen Abbildungen paaren. So konstruierten sie ein beweisbar sicheres Verfahren für iO, das keine neuartigen Annahmen benötigt.

Nach zahlreichen fehlgeschlagenen Versuchen haben Jain und seine Kolleginnen und Kollegen damit gezeigt, dass iO tatsächlich existiert. Allerdings sind die bisherigen Arbeiten rein theoretischer Natur: Selbst die einfachsten Programme würden durch eine solche Obfuskation zu gigantischen Ausmaßen aufgeblasen, was sie in der Praxis völlig unbrauchbar macht. Nach dem Durchbruch bedarf es also weiterer Forschung, um die Methode effizienter zu gestalten – bis dahin ist es aber noch ein langer Weg. ◂

QUELLEN

Ananth, P. et al.: Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. Advances in Cryptology 11694. Springer, Cham, 2019

Barak, B. et al.: On the (im)possibility of obfuscating programs. Advances in Cryptology 2139. Springer, Berlin, Heidelberg, 2001

Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory 22, 1976

Garg, S. et al.: Candidate indistinguishability obfuscation and functional encryption for all circuits. IEEE 54th Annual Symposium on Foundations of Computer Science, Berkeley, CA, USA, 2013

Jain, A. et al.: Indistinguishability obfuscation from well-founded assumptions. Proceedings of the 53nd Annual ACM SIGACT Symposium on Theory of Computing (STOC), 2021